

# Modern Techniques for Implicit Modeling:

ACM SIGGRAPH 2005

Course 13

Course Organizers:

**James F. O'Brien**

University of California at Berkeley

**Terry S. Yoo**

National Library of Medicine, NIH

Lecturers:

**Marc Alexa**

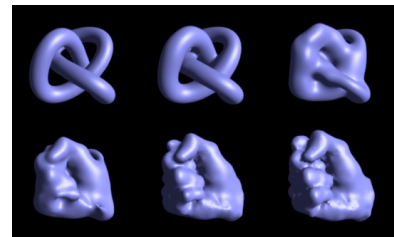
DGM, TU Darmstadt

**Haixia Du**

National Library of Medicine, NIH

**John C. Hart**

University of Illinois Urbana-Champaign



## About the Title Page

**Top figure:** Raycasting view of an implicit surface constructed from the Stanford Bunny model using Compactly Supported Radial Basis Functions (CSRBF). The CSRBF method introduces multiple zero-crossings, a condition that requires some care when raycasting such models. By selecting a transfer function that accents high gradient magnitudes, the desired surface can be rendered. For more information, see the reprint later in these notes, Morse, *et al.*, 2001. Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions, IEEE SMI 2001.

**Middle figure:** Shape transformation using “variational” implicit surfaces. The use of a thin-plate spline as the radial basis function makes possible interpolation between topologically distant objects. The result is smooth transformation between implicit models, including this transition from knot to fist and the corresponding change in genus. For more information, see the reprint: Turk, Greg and James O’Brien, 1999. Shape Transformation Using Variational Implicit Functions, ACM SIGGRAPH 1999.

**Bottom figure:** The Stanford Lucy, consisting of 14 million points, is reconstructed as an MPU implicit with a 0.01% max-norm approximation accuracy; the left part of the model is colored according to the subdivision level which increases from blue to red. The four models in the back are reconstructed from the point set with increasing approximation error. For more information, see the reprint: Yutake Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk and Hans-Peter Seidel, 2003. Multi-level Partition of Unity Implicits, *ACM Transactions on Graphics*, Vol. 22, No. 3, July 2003, Pages 463–470.



# Course Description

This course presents recent developments in modern implicit surfaces, particularly the use of radial-basis functions, MPUs, and digital Morse theory, plus examples of real-world applications from shape transformation to medical modeling. Lectures include the mathematics of implicit modeling and some formal treatment of smoothness issues and sampling constrained implicit surfaces.

## **Course Abstract:**

Several recent advances allow implicit surfaces to move beyond the simple modeling of blobby objects and simple sculpting with constructive geometry. Advanced techniques and emerging methods can now be used for modeling and controlling implicit surfaces generated by approximating and/or interpolating known data points. To demonstrate this, we show how radial basis functions can model various body parts, partial differential equations can be used to mold and shape surfaces, curvature can tie-dye bunnies, and how objects can be extruded from polygon soup. These forms of "constraint-based" or "data-driven" implicit surfaces have begun to supersede previous implicit techniques for modeling objects with biological or natural appearances. Formal approaches to computation, sampling, control, shape transformation, and user-applications will be discussed. In particular, the course will contain new material on the use of nonlinear, partial differential equations in modeling as well as sampling analysis for constrained, interpolating, implicit surfaces. Some people think that implicit surfaces are rubbery, but we will show that they are a solid foundation upon which to build modeling, animation and visualization tools.

This course presents these techniques in a full day of valuable detailed talks, including mathematical foundations of linear algebra, PDEs, sampling, and smoothness, application demonstrations, implementation details and well-documented source code for implementing these techniques. Topics include generating implicit surfaces that interpolate point data, implicit surfaces for shape transformation, surface reconstruction from computer vision data, medical applications, modern level sets, implicit methods to compute medial structures, digital Morse theory, concluding with the presentation of a library of software tools for interactive modeling with implicit surfaces.

## **Prerequisites:**

Attendees should have a good working knowledge of basic graphics techniques and be not easily frightened by terms such as "Partial Differential Equations," "Radial Basis Functions," or "Line Integral." Familiarity with basic implicit surface techniques would be useful, but not necessary.

# Contents

<b>Course Description.....</b>	<b>i</b>
<b>Contents .....</b>	<b>ii</b>
<b>Course Syllabus.....</b>	<b>iv</b>
<b>Speaker Contact Information.....</b>	<b>v</b>
<b>Speaker Biographies .....</b>	<b>vi</b>
<b>Introduction .....</b>	<b>1</b>
“Introduction to Implicit Modeling,” Terry S. Yoo.....	1
<b>Implicit Surfaces that Interpolate .....</b>	<b>13</b>
“Shape Transformation Using Variational Implicit Functions,” Greg Turk and James O'Brien, in <i>Computer Graphics Proceedings, Annual Conference Series</i> (SIGGRAPH 1999), August 1999, pp. 335-342 .....	13
“Modeling with Implicit Surfaces that Interpolate,” Greg Turk and James F. O'Brien, <i>ACM Transactions on Graphics</i> , Vol. 21, No. 4, October 2002, Pages 855–873.....	21
<b>Radial Spline Theory .....</b>	<b>40</b>
“Some Notes on Radial Basis Functions and Thin Plate Splines,” John C. Hart.....	40
“Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling,” Barton T. Stander and John C. Hart, <i>Computer Graphics Proceedings, ACM SIGGRAPH Graphics Proceedings, Annual Conference Series</i> (SIGGRAPH 97), August 1997, pp. 279-286.....	44
“Using the CW-Complex to Represent the Topological Structure of Implicit Surfaces and Solids,” John C. Hart, <i>Proc. Implicit Surfaces '99</i> , Sept. 1999, pp. 107-112.....	52
<b>Compactly Supported RBFs in Implicit Surface Management.....</b>	<b>58</b>
“Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions,” Bryan Morse and Terry S. Yoo and Penny Rheingans and David T. Chen and K.R. Subramanian, <i>Shape Modeling International</i> , Genova, Italy, May 2001.....	78
<b>Implicit Modeling with PDE-based Techniques .....</b>	<b>88</b>
“Implicit Modeling with PDE-based Techniques,” Haixia Du .....	88
“A Shape Design System Using Volumetric Implicit PDEs,” Haixia Du and Hong Qin, <i>Computer-Aided Design</i> 36(2004) 1101-1116. ....	116

<b>Multi-level Partition of Unity Implicits (MPUs) .....</b>	<b>132</b>
“Introduction to Multi-level Partition of Unity Implicits (MPUs),” Marc Alexa.....	132
“Multi-level Partition of Unity Implicits,” Yutake Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk and Hans-Peter Seidel, <i>ACM Transactions on     Graphics</i> , Vol. 22, No. 3, July 2003, Pages 463–470.....	173
<b>Implicit Moving Least Squares Surfaces (IMLS) .....</b>	<b>181</b>
“Interpolating and Approximating Implicit Surfaces from Polygon Soup,” James F. O’Brien. ....	181
“Interpolating and Approximating Implicit Surfaces from Polygon Soup,” Chen Shen, James F. O’Brien, and Jonathan R. Shewchuk, <i>ACM Transactions     on Graphics</i> , Vol. 23, No. 3, August 2004, Pages 896–904. ....	204
“Provably Good Moving Least Squares,” Ravikrishna Kolluri , To Appear in ACM-SIAM Symposium on Discrete Algorithms 2005.....	213
<b>Medical Applications of Implicit Surfaces .....</b>	<b>223</b>
“Medical Applications of Implicit Surfaces,” Terry S. Yoo. ....	223
“Anatomic Modeling from Unstructured Samples Using Variational Implicit Surfaces,” Terry S. Yoo, Bryan Morse, K.R. Subramanian, Penny Rheingans, and Michael J. Ackerman, In <i>Studies in Health Technology and Informatics</i> , vol. 81, (Proceedings of Medicine Meets Virtual Reality 2001. J. D. Westwood, <i>et al.</i> , eds.), Amsterdam: IOS Press, pp. 594-600. ....	245
“Active Contours Using a Constraint-Based Implicit Representation,” Bryan Morse, Weiming Liu, Terry S. Yoo, and K.R. Subramanian, To appear in <i>Proceedings Computer Vision and Pattern Recognition</i> , IEEE Computer Society Press, June 2005. ....	252
<b>Wickbert: An Open-Source Interactive Implicit Surface Modeler .....</b>	<b>260</b>
“Using Particles To Sample And Control Implicit Surfaces,” Andrew P. Witkin and Paul S. Heckbert, <i>Computer Graphics Proceedings, ACM     SIGGRAPH Graphics Proceedings, Annual Conference Series (SIGGRAPH     94)</i> , July 1994, pp. 269 - 277.....	260
“Using Particles to Sample and Control More Complex Implicit Surfaces,” John Hart, Ed Bachta, Wojciech Jarosz and Terry Fleury, in <i>Proceedings of     Shape Modeling International 2002</i> . ....	269
“A Programmable Particle System Framework For Shape Modeling,” Wen Y. Su and John Hart, in <i>Proceedings of Shape Modeling International</i> June 2005.....	277

## Course Syllabus

8:30 - Yoo

Welcome, Introduction to Implicit Modeling (25 min)

8:55 - O'Brien

Implicit Surfaces that Interpolate (35 min)

9:30 - Hart

Radial Spline Theory (45 min)

•10:15 - Break

10:30 - Yoo

Compactly Supported RBFs in Implicit Surface Management (55 min)

11:25 - Du

Implicit Modeling with PDE-based Techniques (50 min)

•12:15 - Lunch

1:45 - Alexa

Multi-level Partition of Unity Implicits (MPUs) (55 min)

2:40 - O'Brien

Implicit Moving Least Squares Surfaces (IMLS) (50 min)

•3:30 - Break

3:45 - Yoo

Medical Applications of Implicit Surfaces (45 min)

4:30 - Hart

Wickbert: An Open-Source Interactive Implicit Surface Modeler (60 min)

•5:30 - End

## Speaker Contact Information

### **Marc Alexa**

Darmstadt University of Technology  
Discrete Geometric Modeling Group  
Fraunhoferstr. 5  
D-64283 Darmstadt

Phone: +49 6151 155 679

Fax: +49 6151 155 669

[alexa@informatik.tu-darmstadt.de](mailto:alexa@informatik.tu-darmstadt.de)

### **Haixia Du**

Visiting Scientist  
National Library of Medicine  
National Institutes of Health  
8600 Rockville Pike  
Bethesda, MD 20894

(301) 435-3268

fax: (301) 402-4080

[hdu@mail.nih.gov](mailto:hdu@mail.nih.gov)

### **John C. Hart**

Associate Professor  
Department of Computer Science  
University of Illinois Urbana-  
Champaign  
3212 DCL, MC 258  
1304 W Springfield  
Urbana, IL 61801

(217) 333-8740

fax: (217) 244-6869

[jch@cs.uiuc.edu](mailto:jch@cs.uiuc.edu)

### **James F. O'Brien**

Assistant Professor  
EECS, Computer Science Division  
633 Soda Hall, Mail Code 1776  
University of California at Berkeley  
Berkeley, California 94720-1776

(510) 642-0865

fax: (510) 642-5775

[job@eecs.berkeley.edu](mailto:job@eecs.berkeley.edu)

### **Terry S. Yoo**

Head, 3D Informatics Program  
National Library of Medicine  
National Institutes of Health  
8600 Rockville Pike  
Bethesda, MD 20894

(301) 435-3268

fax: (301) 402-4080

[yoo@nlm.nih.gov](mailto:yoo@nlm.nih.gov)

## Speaker Biographies

**Marc Alexa** is an Assistant Professor of Computer Science at Darmstadt University of Technology and heads the Discrete Geometric Modeling group. He is interested in representing shapes and their deformation, using point sampled geometry, implicit surfaces, explicit representations, and linear spaces of base shapes. He has lectured on topics related to shape representations at SIGGRAPH and other conferences, has been a co-chair and has served as a member of several committees of major graphics conferences, and will be papers co-chair of Eurographics 2005 and general co-chair of the ACM/Eurographics Symposium on Point Based Rendering 2005.

**Haixia Du** is a Postdoctoral Fellow in the Office of High Performance Computing and Communications at the National Library of Medicine in the 3D Informatics Group. Her research interests are in geometric and physics-based modeling, visualization, and medical imaging, with emphasis on PDE-based shape modeling, including shape design, reconstruction, metamorphosis, and simplification. She has co-authored several papers on implicit surface reconstruction and manipulation from curve sketches, scattered data points, and volumetric data using PDE techniques. Haixia received her Ph.D. in Computer Science from Stony Brook University in 2004.

**John C. Hart** is Associate Professor of Computer Science at the University of Illinois Urbana-Champaign. In 1993 he received an NSF award to explore implicit surfaces, and got hooked. He co-chaired the 1996 Eurographics/SIGGRAPH Workshop on Implicit Surfaces, and has organized/lectured in previous SIGGRAPH courses, including several on implicit surfaces. Hart is co-author of *Real-Time Shading* and a contributing author of *Modeling and Texturing: A Procedural Approach*, 3<sup>rd</sup> edition. Hart is the Editor-in-Chief of ACM Transactions on Graphics. He served five years on the SIGGRAPH Executive Committee and was an executive producer of the documentary "The Story of Computer Graphics."

**James F. O'Brien** is an Assistant Professor of Computer Science at the University of California, Berkeley. His interests focus on generating realistic motion using physically based simulation and motion-capture techniques. He has authored several papers on these topics, including ten presented at SIGGRAPH and his work has been featured multiple times in SIGGRAPH's Electronic Theater. He received his doctorate from the Georgia Institute of Technology in 2000, the same year he joined Berkeley's Faculty. O'Brien is a Sloan Fellow, Technology Review selected him one of their TR-100 for 2004, and he was recently awarded grants from the Okawa and Hellman Foundations.

**Terry S. Yoo** is a Computer Scientist in the Office of High Performance Computing and Communications, National Library of Medicine, NIH, where he explores the processing and visualizing of 3D medical data, interactive 3D graphics, and computational geometry. Previously as a professor of Radiology, he managed a research program in Interventional MRI with the University of Mississippi. Terry holds an A.B. in Biology from Harvard, and a M.S. and Ph.D. in Computer Science from UNC Chapel Hill.



# **SIGGRAPH2005**



**SIGGRAPH2005**

---

## **Modern Techniques for Implicit Modeling**

**James F. O'Brien**

University of California Berkeley

**Terry S. Yoo**

National Library of Medicine, NIH

## Why implicit surfaces?



- Implicit surfaces are as old as graphics...
  - A compact representation for orientable, closed surfaces.
  - Mathematically tractable:
    - Can control mathematical continuity of the surface.
    - No cracks.
  - Not a basic machine representation
    - Not polygons, not textures, not voxels...

## Why this course? ...



- Implicit surfaces have been:
  - Capricious, generating spurious nodes/blobs
  - Inclined to make creases, folds, and pinches
  - Difficult to craft surfaces that conform to physical models
- Recent advances show promise for:
  - Character animation, shape transformation
  - Medical modeling, computer vision
  - Topology control, front evolution (level sets)



## Modern Techniques for Implicit Modeling...



- We will cover implicit surfaces of/by/for:
  - Shape Transformation
  - Medical Applications
  - Computer Vision
  - Digital Morse Theory
  - Software
  - Level Sets
  - Level Set Applications
- We will not cover:
  - Comprehensive Introduction to implicit surfaces
  - Blob Trees
  - Parametric surfaces
  - Meta-balls
  - Constructive geometry

## Lecturers



Marc Alexa

- DGM, TU Darmstadt

Haixia Du

- NLM / NIH

John Hart

- Univ. of Illinois Urbana-Champaign

James F. O'Brien

- Georgia Institute of Technology

Terry S. Yoo

- NLM / NIH

## Course at a glance



8:30 - Yoo  
Welcome, Introduction to Implicit Modeling

8:55 - O'Brien  
Implicit Surfaces that Interpolate

9:30 - Hart  
Radial Spline Theory

- 10:15 - Break

10:30 - Yoo  
Compactly Supported RBFs in Implicit Surface Management

11:25 - Du  
Implicit Modeling with PDE-based Techniques

- 12:15 - Lunch

1:45 - Alexa  
Multi-level Partition of Unity Implicits (MPUs)

2:40 - O'Brien  
Implicit Moving Least Squares Surfaces (IMLS)

- 3:30 - Break

3:45 - Yoo  
Medical Applications of Implicit Surfaces

4:30 - Hart  
Wickbert: An Open-Source Interactive Implicit Surface Modeler

- 5:30 - End

## Course Supplementary Materials Website



- Links

<http://visual.nlm.nih.gov/tutorials/sig2005/index.html>

<http://graphics.cs.uiuc.edu/projects/surface>

- Text

J. Bloomenthal, ed. Introduction to Implicit Surfaces.  
Morgan Kaufmann. 1997. pp. 332.



## Introduction to Implicit Modeling

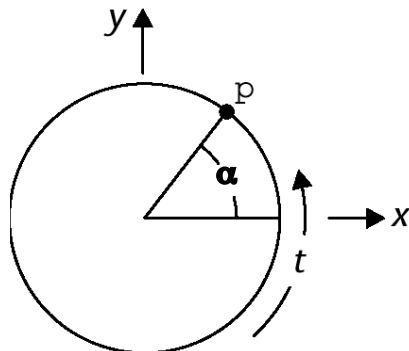
Terry S. Yoo

National Library of Medicine, NIH



## Some SIGGRAPHistory

- Blobs and Metaballs
  - J.Blinn, A Generalization of Algebraic Surface Drawing, ACM Trans. Graphics, 1:3, Jul. 1982, pp. 135-256.
  - S.Muraki, Volumetric Shape Description of Range Data Using Blobby Model, Computer Graphics, 25:4, 1991, pp. 227-235 (Proc. SIGGRAPH 91).
- Constructive solid geometry (polyhedral & implicit surfaces)
  - D. Laidlaw, W. Trumbore and J. Hughes. Constructive Solid Geometry for Polyhedral Objects. In Computer Graphics, Proceedings of SIGGRAPH '86, pages 161-170, August 1986.
  - G.Wyvill, C.McPheeters, B.Wyvill, Data Structure for Soft Objects, The Visual Computer, 2:4, Aug. 1986, pp. 227-234.



### Equiangular Parametric

(transcendental trigonometric)

$$p = (\cos(\alpha), \sin(\alpha))$$

### Non-Equiangular Parametric

(rational trigonometric)

$$p = (\pm(1-t^2) / (1+t^2), 2t / (1+t^2))$$

### Implicit

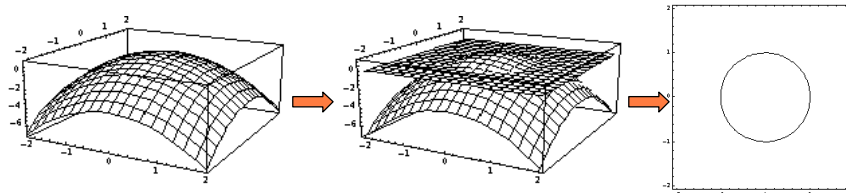
$$p_x^2 + p_y^2 - 1 = 0$$

## Implicit Surfaces

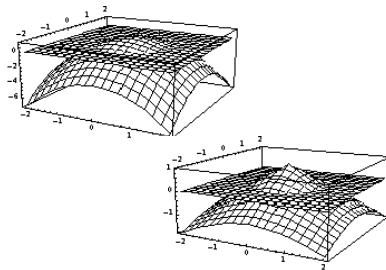


SIGGRAPH2005

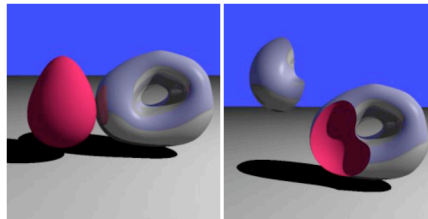
- Not patches (no seams).
- Oriented surfaces (have a natural inside and outside).
- Differentiable, closed, Continuous



## Implicit Surfaces

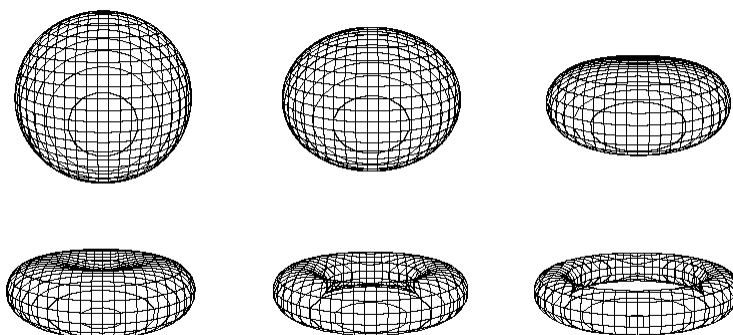


- No origin on the manifold.
- Usually, no unique solutions.
- Can combine functions, apply functions from constructive solid geometry.

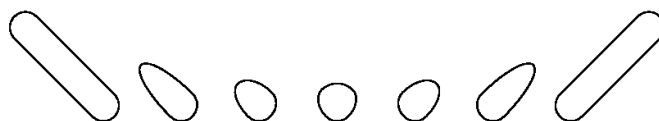


**Polygonized CSG objects.**  
courtesy Brian Wyvill  
and Kees van Overveld

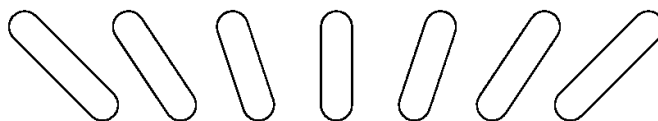
## Natural Shape Transformation



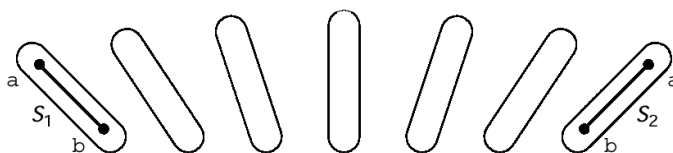
$f(p) = td(S_1, p) + (1 - t)d(S_2, p) - c$ , where  $d(S, p)$  is Euclidean distance between point  $p$  and segment  $S$



$f(p) = td^2(S_1, p) + (1 - t)d^2(S_2, p) - c$

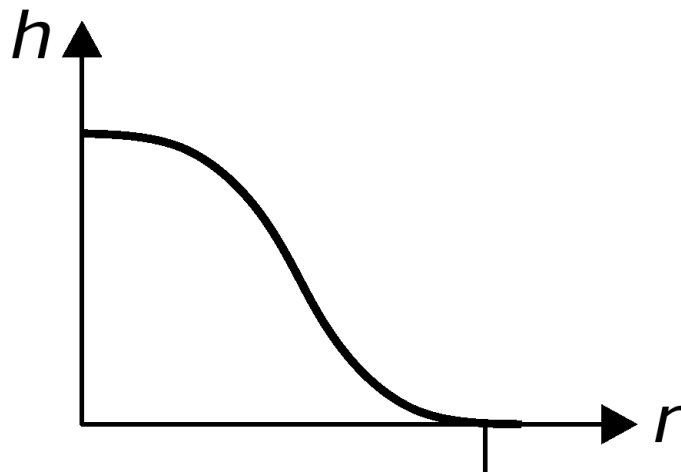


$f(p) = d(S, p) - c$ , where  $S = tS_1 + (1 - t)S_2$

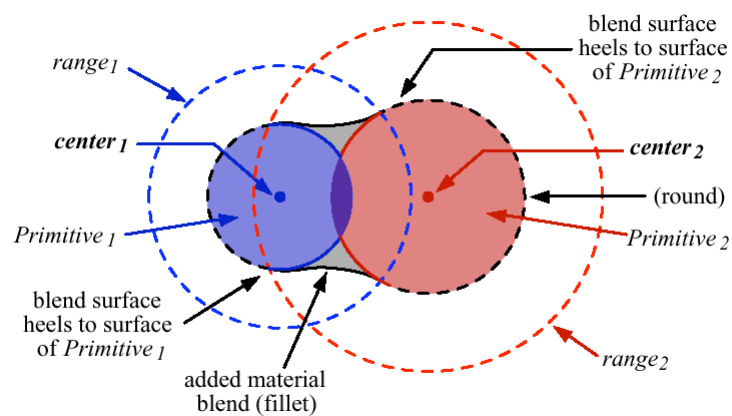


$f(p) = d(S, p) - c$ , where  $S$  is a rigid body rotation between  $S_1$  and  $S_2$

## Compactly Supported Implicit Characteristic Fcn



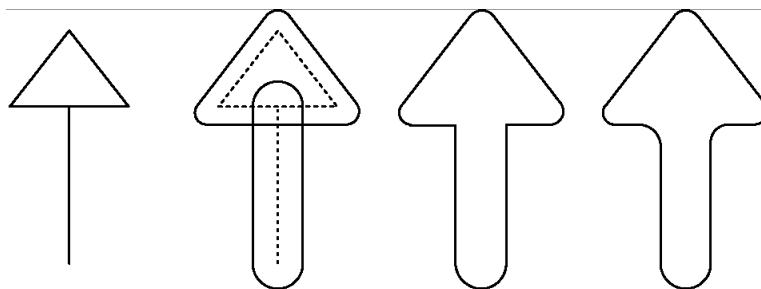
## Natural Blending



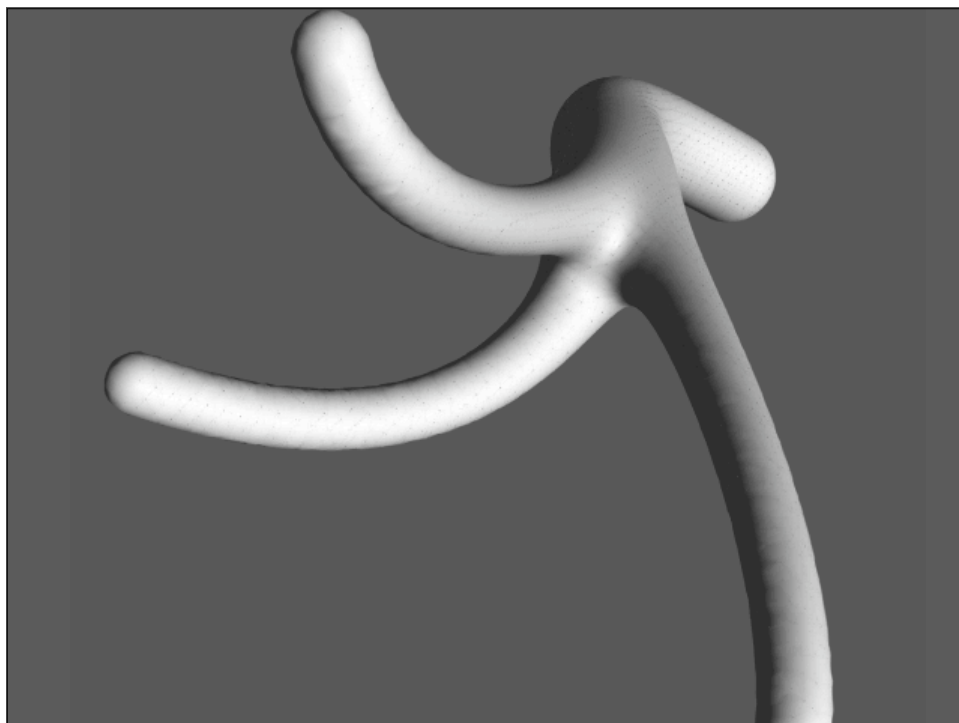
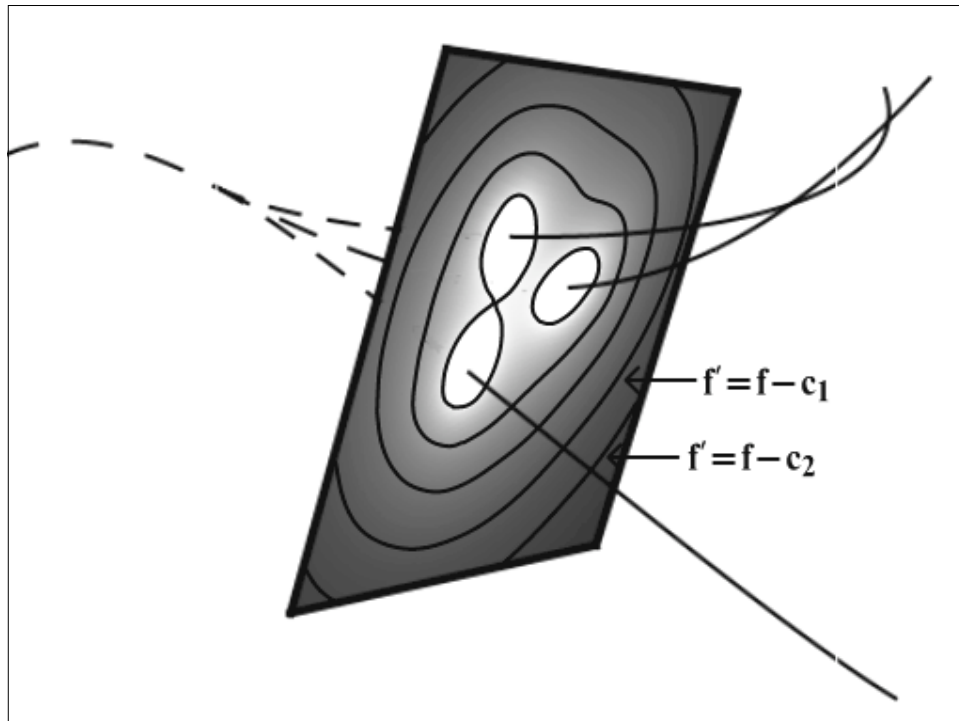
## More SIGGRAPH History



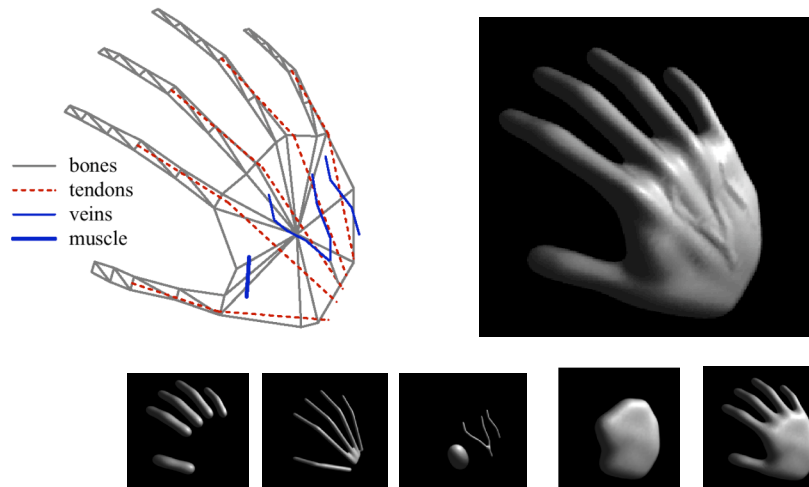
- Generative Models: Implicit, Parametric, Medical
  - J. Bloomenthal, K. Shoemake, Convolution Surfaces, Computer Graphics 25:4, 1991 (Proc. SIGGRAPH), pp. 251-256.
  - Snyder, John M. and James T. Kajiya, "Generative modeling: a symbolic system for geometric modeling," [Figures], in Proceedings of SIGGRAPH 1992, ACM SIGGRAPH, 1992, pp. 369-378.
  - Joshi, S., SM Pizer, PT Fletcher, A Thall, G Tracton, Multi-Scale 3-D Deformable Model Segmentation Based on Medical Description. Information Processing in Medical Imaging (IPMI 2001), MF Insana, RM Leahy, eds., Springer LNCS 2082: 64-77.
- Level Sets (discrete implicit surfaces)
  - J. Sethian. 1999. Level Set Methods and Fast Marching Methods, second ed. Cambridge University Press, Cambridge, UK.
  - Ross T. Whitaker, A Level-Set Approach to 3D Reconstruction from Range Data, International Journal of Computer Vision, v.29 n.3, p.203-231, Sept. 1998
  - K. Museth, D.E. Breen, R.T. Whitaker and A.H. Barr, "Level Set Surface Editing Operators," ACM Transactions on Graphics (Proc. SIGGRAPH), Vol. 21, No. 3, July 2002, pp. 330-338.



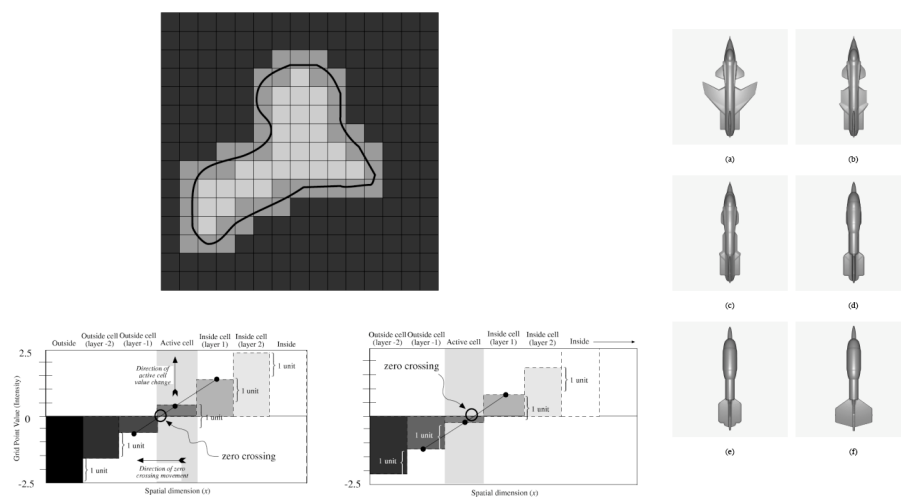


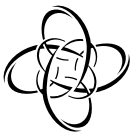


## Convolution Surfaces



## Level Sets





# Shape Transformation Using Variational Implicit Functions

Greg Turk

James F. O'Brien

Georgia Institute of Technology

## Abstract

Traditionally, shape transformation using implicit functions is performed in two distinct steps: 1) creating two implicit functions, and 2) interpolating between these two functions. We present a new shape transformation method that combines these two tasks into a single step. We create a transformation between two  $N$ -dimensional objects by casting this as a scattered data interpolation problem in  $N + 1$  dimensions. For the case of 2D shapes, we place all of our data constraints within two planes, one for each shape. These planes are placed parallel to one another in 3D. Zero-valued constraints specify the locations of shape boundaries and positive-valued constraints are placed along the normal direction in towards the center of the shape. We then invoke a variational interpolation technique (the 3D generalization of thin-plate interpolation), and this yields a single implicit function in 3D. Intermediate shapes are simply the zero-valued contours of 2D slices through this 3D function. Shape transformation between 3D shapes can be performed similarly by solving a 4D interpolation problem. To our knowledge, ours is the first shape transformation method to unify the tasks of implicit function creation and interpolation. The transformations produced by this method appear smooth and natural, even between objects of differing topologies. If desired, one or more additional shapes may be introduced that influence the intermediate shapes in a sequence. Our method can also reconstruct surfaces from multiple slices that are not restricted to being parallel to one another.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surfaces and object representations

**Keywords:** Shape transformation, shape morphing, contour interpolation, implicit surfaces, thin-plate techniques.

## 1 Introduction

The shape transformation problem can be stated as follows: Given two shapes A and B, construct a sequence of intermediate shapes so that adjacent pairs in the sequence are geometrically close to one another. Playing the resulting sequence of shapes as an animation would show object A deforming into object B. Sequences of 2D shapes can be thought of as slices through a 3D surface, as shown in Figure 1. Shape transformation can be performed between objects of any dimension, although 2D and 3D shapes are by far the most common cases. Shape transformation has applications in medicine, computer aided design, and special effects creation. We give an overview of these three applications below.

One important application of shape transformation in medicine is contour interpolation. Non-invasive imaging techniques often col-

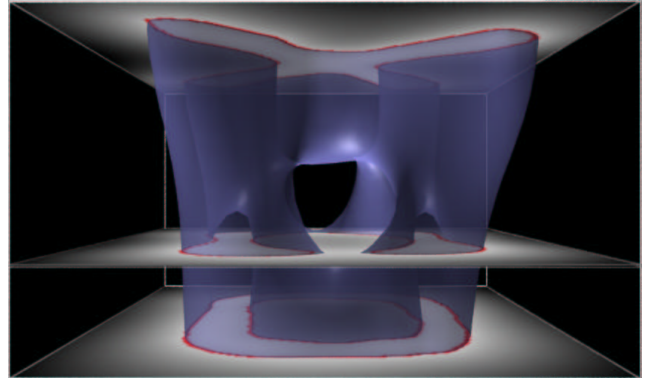


Figure 1: Visualization of transformation between X and O shapes. Top and bottom planes contain constraints for the two shapes. Translucent surface is the isosurface of a 3D variational implicit function, and slices through it give intermediate shapes.

lect data about a patient's internal anatomy in "slices" of a particular size such as  $512 \times 512$  samples. Usually many fewer slices are taken along the third dimension so that a resulting volume might, for example, be sampled at  $512 \times 512 \times 30$  resolution. To reconstruct a 3D model of a particular organ, the samples are segmented to create shapes (contours) within the slices. Intermediate shapes are then created between slices in the sparsely sampled dimension. The reconstructed 3D object is formed by stacking together the original and the interpolated contours. This is an example of 2D shape transformation.

Shape transformation can also be a useful tool in computer aided geometric design. Consider the problem of creating a join between two metal parts with different cross-sections. It is important for the connecting surface to be smooth because those places with sharp ridges or creases are the locations that are most likely to form cracks. The intermediate surface joining the two parts can be created using shape transformation, much in the same way as with contour interpolation for medical imaging. Because of the smoothness properties of variational interpolation methods, we consider them a natural tool to explore for shape transformation in CAD.

Finally, animated shape transformations have been used to create dramatic special effects for feature films and commercials. One of the best-known examples of shape transformation is in the film *Terminator 2*. In this film, a cyborg policeman undergoes a number of transformations from an amorphous and highly reflective surface to various destination shapes. 2D image morphing would not have accurately modeled the reflection of the environment off the surface of the deforming cyborg, hence tailor-made 3D shape transformation programs were used for these effects [9].

In this paper we use variational interpolation in a new way to produce high-quality shape transformations that may be used for any of the previously mentioned applications. Our method allows a user to control the transformation in several ways, and it is general enough to produce transformations between shapes of any topology.

## 2 Previous Work

Most shape transformation techniques can be placed into one of two categories: parametric correspondence methods and implicit

turk@cc.gatech.edu, job@acm.org.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SIGGRAPH 99, Los Angeles, CA USA

Copyright ACM 1999 0-201-48560-5/99/08 . . . \$5.00

function interpolation. Parametric methods are typically faster to compute and require less memory because they operate on a lower-dimensional representation of an object than do implicit function methods. Unfortunately, transforming between objects of different topologies is considerably more difficult with parametric methods. Parametric approaches also can suffer from problems with self-intersecting surfaces, but this is never a problem with implicit function methods. Techniques that use implicit function interpolation gracefully handle changes in topology between objects and do not create self-intersecting surfaces.

A parametric correspondence approach to shape transformation attempts to find a “reasonable” correspondence between pairs of locations on the boundaries of the two shapes. Intermediate shapes are then created by computing interpolated positions between the corresponding pairs of points. Many shape transformation techniques have been created that follow the parametric correspondence approach. One early application of this approach is the method of contour interpolation described by Fuchs, Kedem and Uselton [10]. Their method attempts to find an “optimal” (minimum-area) triangular tiling that connects two contours using dynamic programming. Many subsequent techniques followed this approach of defining a quality measure for a particular correspondence between contours and then invoking an optimization procedure [22, 25]. There have been fewer examples of using parametric correspondence for 3D shape transformation. One quite successful 3D parametric method is the work of Kent et al. [17]. The key to their approach is to subdivide the polygons of the two models in a manner that creates a correspondence between the vertices of the two models. More recently, Gregory and co-workers created a similar method that also allows a user to specify region correspondences between meshes to better control a transformation [12].

An entirely different approach to shape transformation is to create an implicit function for each shape and then to smoothly interpolate between these two functions. A shape is defined by an implicit function,  $f(\mathbf{x})$ , as the set of all points  $\mathbf{x}$  such that  $f(\mathbf{x}) = 0$ . For contour interpolation in 2D, the implicit function can be thought of as a height field over a two-dimensional domain, and the boundary of a shape is the one-dimensional curve defined by all the points that have the same elevation value of zero. An implicit function in 3D is a function that yields a scalar value at every point in 3D. The shape described by such a function is given by those places in 3D whose function value is zero (the isosurface).

One commonly used implicit function is the *inside/outside function* or *characteristic function*. This function takes on only two values over the entire domain. The two values that are typically used are zero to represent locations that are outside and one to signify positions that are inside the given shape. Given a powerful enough interpolation technique, the characteristic function can be used for creating shape transformations. Hughes presented a successful example of this approach by transforming characteristic functions into the frequency domain and performing interpolation on the frequency representations of the shapes [15]. Kaul and Rossignac found that smooth intermediate shapes can be generated by using weighted Minkowski sums to interpolate between characteristic functions [16]. They later created a generalization of this technique that can use several intermediate shapes to control the interpolation between objects [24]. Using a wavelet decomposition of a characteristic function allowed He and colleagues to create intermediates between quite complex 3D objects [13].

A more informative implicit function can provide excellent intermediate shapes even if a simple interpolation technique is used. In particular, the *signed distance function* (sometimes called the *distance transform*) is an implicit function that gives very plausible intermediate shapes even when used with simple linear interpolation of the function values of the two shapes. The value of the signed distance function at a point  $\mathbf{x}$  inside a given shape is just the Euclidean distance between  $\mathbf{x}$  and the nearest point on the boundary of the shape. For a point  $\mathbf{x}$  that is outside the shape, the signed distance function takes on the negative of the distance from  $\mathbf{x}$  to the closest point on the boundary.

Several researchers have used the signed distance function to interpolate between 2D contours [19, 14]. The distance function for each given shape is represented as a regular 2D grid of values, and an intermediate implicit function is created by linear interpolation between corresponding grid values of the two implicit functions. Each intermediate shape is given by the zero iso-contour of this interpolated implicit function. In contrast to the global interpolation methods described above (frequency domain, wavelets, Minkowski sum), this interpolation is entirely local in nature. Nevertheless, the shape transformations that are created by this method are quite good. In essence, the information that the signed distance function encodes (distance to nearest boundary) is enough to make up for the purely local method of interpolation. Payne and Toga were the first to transform three dimensional shapes using this approach [23]. Cohen-Or and colleagues gave additional control to this same approach by combining it with a warping technique in order to produce shape transformations of 3D objects [7].

Our approach to shape transformation combines the two steps of building implicit functions and interpolating between them. To our knowledge, it is the only method to do so. The remainder of this paper describes how variational interpolation can be used to simultaneously solve these two tasks.

### 3 Variational Interpolation

Our approach relies on *scattered data interpolation* to solve the shape transformation problem. The problem of scattered interpolation is to create a smooth function that passes through a given set of data points. The two-dimensional version of this problem can be stated as follows: Given a collection of  $k$  constraint points  $\{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k\}$  that are scattered in the plane, together with scalar height values at each of these points  $\{h_1, h_2, \dots, h_k\}$ , construct a smooth surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function  $f(\mathbf{x})$  so that  $f(\mathbf{c}_i) = h_i$ , for  $1 \leq i \leq k$ .

One common approach to solving scattered data problems is to use variational techniques (from the calculus of variations). This approach begins with an energy that measures the quality of an interpolating function and then finds the single function that matches the given data points and that minimizes this energy measure. For two-dimensional problems, thin-plate interpolation is the variational solution when using the following energy function  $E$ :

$$E = \int_{\Omega} f_{xx}^2(\mathbf{x}) + 2f_{xy}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x}) \quad (1)$$

The notation  $f_{xx}$  means the second partial derivative in the  $x$  direction, and the other two terms are similar partial derivatives, one of them mixed. The above energy function is basically a measure of the aggregate squared curvature of  $f(\mathbf{x})$  over the region of interest  $\Omega$ . Any creases or pinches in a surface will result in a larger value of  $E$ . A smooth surface that has no such regions of high curvature will have a lower value of  $E$ . The thin-plate solution to an interpolation problem is the function  $f(\mathbf{x})$  that satisfies all of the constraints and that has the smallest possible value of  $E$ .

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points  $\mathbf{c}_i$  are positions in  $N$ -dimensions rather than in 2D, this is called the  $N$ -dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for other dimensions. In this paper we will perform interpolation in two, three, four and five dimensions. Because the term *thin-plate* is only meaningful for 2D problems, we will use *variational interpolation* to mean the generalization of thin-plate techniques to any number of dimensions.

The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function,  $f(\mathbf{x})$ , that will minimize equation 1 subject to the interpolation constraints  $f(\mathbf{c}_i) = h_i$ . Equation 1 can be solved using weighted sums of the

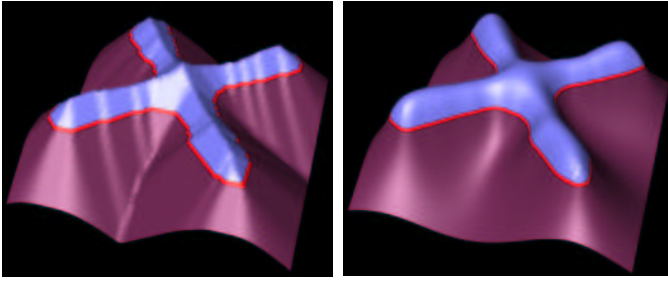


Figure 2: Implicit functions for an X shape. Left shows the signed distance function, and right shows the smoother variational implicit function.

radial basis function  $\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|)$ . The family of variational problems that includes equation 1 was studied by Duchon [8].

Using the appropriate radial basis function, we can then express the interpolation function as

$$f(\mathbf{x}) = \sum_{j=1}^n d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \quad (2)$$

In the above equation,  $\mathbf{c}_j$  are the locations of the constraints, the  $d_j$  are the weights, and  $P(\mathbf{x})$  is a degree one polynomial that accounts for the linear and constant portions of  $f$ . Because the thin-plate radial basis function naturally minimizes equation 1, determining the weights,  $d_j$ , and the coefficients of  $P(\mathbf{x})$  so that the interpolation constraints are satisfied will yield the desired solution that minimizes equation 1 subject to the constraints. Furthermore, the solution will be an exact analytic solution, and is not subject to approximation and discretization errors that may occur when using finite element or finite difference methods.

To solve for the set of  $d_j$  that will satisfy the interpolation constraints  $h_i = f(\mathbf{c}_i)$ , we can substitute the right side of equation 2 for  $f(\mathbf{c}_i)$ , which gives:

$$h_i = \sum_{j=1}^k d_j \phi(\mathbf{c}_i - \mathbf{c}_j) + P(\mathbf{c}_i) \quad (3)$$

Since this equation is linear with respect to the unknowns,  $d_j$  and the coefficients of  $P(\mathbf{x})$ , it can be formulated as a linear system. For interpolation in 3D, let  $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$  and let  $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$ . Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \dots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The above system is symmetric and positive semi-definite, so there will always be a unique solution for the  $d_j$  and  $p_j$  [11]. For systems with up to a few thousand constraints, the system can be solved directly with a technique such as symmetric LU decomposition. We used symmetric LU decomposition to solve this system for all of the examples shown in this paper.

Using the tools of variational interpolation we can now turn our attention to creating implicit functions for shape transformation.

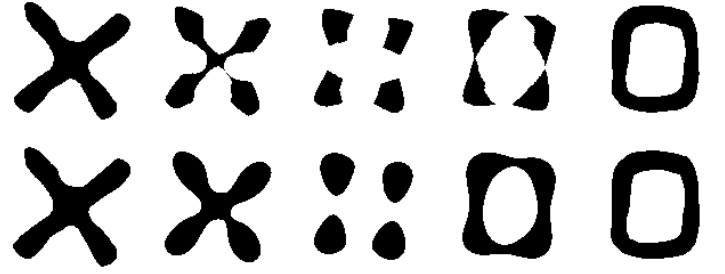


Figure 3: Upper row is a shape transformation created using the signed distance transform. Lower row is the sequence generated using a single variational implicit function.

## 4 Smooth Implicit Function Creation

In this section we will lay down the groundwork for shape transformation by discussing the creation of smooth implicit functions for a single shape. In particular, we will use variational interpolation of scattered constraints to construct implicit functions. Later we will generalize this to create functions that perform shape transformation.

Let us first examine the signed distance transformation because it is commonly used for shape transformation. The left half of Figure 2 shows a height field representation of the signed distance function of an X shape. The figure shows sharp ridges (the *medial axis*) that run down the middle of the height field. Ridges appear in the middle of shapes where the points are equally distant from two or more boundary points of the original shape. The values of a signed distance function decrease as one moves away from the ridge towards the boundaries. Figure 3, top row, shows a shape interpolation sequence between an X and an O shape that was created by linear interpolation between two signed distance functions. Note the pinched portions of some of the intermediate shapes. These sharp features are not isolated problems, but instead persist over many intermediate shapes. The cause of these pinches are the sharp ridges of signed distance functions. In many applications such artifacts are undesirable. In medical reconstruction, for example, these pinches are a poor estimate of shape because most biological structures have smooth surfaces. Because of this, we seek implicit functions that are continuous and that have a continuous first derivative.

### 4.1 Variational Implicit Functions in 2D

We can create smooth implicit functions for a given shape using variational interpolation. This can be done both for 2D and 3D shapes, although we will begin by discussing the 2D case. In this approach, we create a closed 2D curve by describing a number of locations through which the curve will pass and also specifying a number of points that should be interior to the curve. We call the given points on the curve the *boundary constraints*. The boundary constraints are locations at which we require our implicit function to take on the value of zero. Paired with each boundary constraint is a *normal constraint*, which is a location at which the implicit function is required to take on the value one. (Actually, any positive value could be used.) The locations of the normal constraints should be towards the interior of the desired curve, and also the line passing through the normal constraint and its paired boundary constraint should be parallel to the desired normal to the curve. The collection of boundary and normal constraints are passed along to a variational interpolation routine as the scattered constraints to be interpolated. The function that is returned is an implicit function that describes our curve. The curve will exactly pass through our boundary constraints.

Figure 4 (left) illustrates eight such pairs of constraints in the plane, with the boundary constraints shown as circles and the normal constraints as plus signs. When we invoke variational interpo-



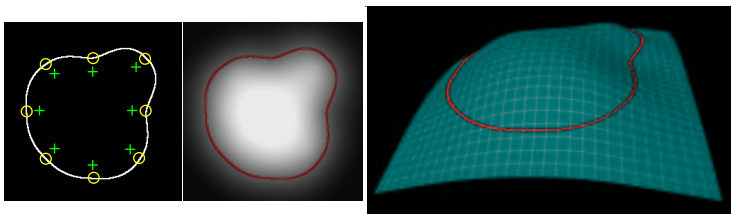


Figure 4: At left are pairs of boundary and normal constraints (circles and pluses). The middle image uses intensity to show the resulting variational implicit function, and the right image shows the function as a height field.

lation with such constraints, the result is a function that takes on the value of zero exactly at our zero-value constraints and that is positive in the direction of our normal constraints (towards the interior of the shape). The closed curve passing through the zero-value constraints in Figure 4 (middle) is the iso-contour of the implicit function created by this method. Figure 4 (right) shows the resulting implicit function rendered as a height field. Given enough suitably-placed boundary constraints we can define any closed shape. We call an implicit function that is created in this manner a *variational implicit function*. This new technique for creating implicit functions also show promise for surface modeling, a topic that is explored in [27].

We now turn our attention to defining boundary and normal constraints for a given 2D shape. Assume that a given shape is represented as a gray-scale image. White pixels represent the interior of a shape, black pixels will be outside the shape, and pixels with intermediate gray values lie on the boundary of the shape. Let  $m$  be the middle gray value of our image's gray scale range. Our goal is to create constraints between any two adjacent pixels where one pixel's value is less than  $m$  and the other's value is greater. Identifying these locations is the 2D analog of finding the vertex locations in a 3D marching cubes algorithm [21].

We traverse the entire gray-scale image and examine the east and south neighbor of each pixel  $I(x, y)$ . If  $I(x, y) < m$  and either neighbor has a value greater than  $m$ , we create a boundary constraint at a point along the line segment joining the pixel centers. A boundary constraint is also created if  $I(x, y) > m$  and either neighbor takes on a value less than  $m$ . The value of the constraint is zero, and we set the position of the constraint at the location between the two pixels where the image would take on the value of  $m$  if we assume linear interpolation of pixel values. Next, we estimate the gradient of the gray scale image using linear interpolation of pixel values and central differencing. We then create a normal constraint a short distance away from the zero crossing in the direction of the gradient. We have found that a distance of a pixel's width between the boundary and normal constraints works well in practice. Figure 2 (right) shows the implicit function for an X shape that was created using variational interpolation from such constraints. It is smooth and free of sharp ridges.

## 4.2 Variational Implicit Functions in 3D

We can create implicit functions for 3D surfaces using variational interpolation in much the same way as for 2D shapes. Specifically, we can derive 3D constraints from the vertex positions and surface normals of a polygonal representation of an object. Let  $(x, y, z)$  and  $(n_x, n_y, n_z)$  be the position and the surface normal at a vertex, respectively. Then a boundary constraint is placed at  $(x, y, z)$  and a normal constraint is placed at  $(x - kn_x, y - kn_y, z - kn_z)$ , where  $k$  is some small value. We use a value of  $k = 0.01$  for models that fit within a unit cube for the results shown in this paper. All of the 3D models that we transform in this paper were constructed by building an implicit function in this manner. Note that we can use this method to build an implicit function whenever we have a collection of points and normals—polygon connectivity is not necessary.

Now that we can construct smooth implicit functions for both two- and three-dimensional shapes, we turn our attention to shape transformation. It would be possible to create variational implicit functions for each of two given shapes and then linearly interpolate between these functions to create a shape transformation sequence. Instead, however, we will examine an even better way of performing shape transformation by generalizing the implicit function building methods of this section.

## 5 Unifying Function Creation and Interpolation

The key to our shape transformation approach is to represent the entire sequence of shapes with a single implicit function. To do so, we need to work in one higher dimension than the given shapes. For 2D shapes, we will construct an implicit function in 3D that represents our two given shapes in two distinct parallel planes. This is actually simple to achieve now that we know how to use scattered data interpolation to create an implicit function.

### 5.1 Two-Dimensional Shape Transformation

Given two shapes in the plane, assume that we have created a set of boundary and normal constraints for each shape, as described in Section 4. Instead of using each set of constraints separately to create two different 2D implicit functions, we will embed all of the constraints in 3D. We do this by adding a third coordinate value to the location of each boundary and normal constraint. For those constraints for the first shape, we set the new coordinate  $t$  for all constraints to  $t = 0$ . For the second shape, all of the new coordinate values are set to  $t = t_{max}$  (some non-zero value). Although we have added a third dimension to the locations of the constraints, the values that are to be interpolated remain unchanged for all constraints.

Once we have placed the constraints of both shapes into 3D, we invoke 3D variational interpolation to create a single scalar-valued function over  $\mathbf{R}^3$ . If we take a slice of this function in the plane  $t = 0$ , we find an implicit function that takes on the value zero exactly at the boundary constraints for our first shape. In this plane, our function describes the first shape. Similarly, in the plane  $t = t_{max}$  this function gives the second shape. Parallel slices at locations between these two planes ( $0 < t < t_{max}$ ) represent the shapes of our shape transformation sequence. Figure 1 illustrates that the collection of intermediate shapes are all just slices through a surface in 3D that is created by variational interpolation.

Figure 3 (bottom) shows the sequence of shapes created using this variational approach to shape transformation. Topology changes (e.g. the addition or removal of holes) come “for free”, without any human guidance or algorithmic complications. Notice that all of the intermediate shapes have smooth boundaries, without pinches. Sharp features can arise only momentarily when there is a change in topology such as when two parts join. Figure 5 shows two more shape transformations that use this approach and that also incorporate warping. Warping is another degree of control that may be added to any shape transformation technique, and is in fact

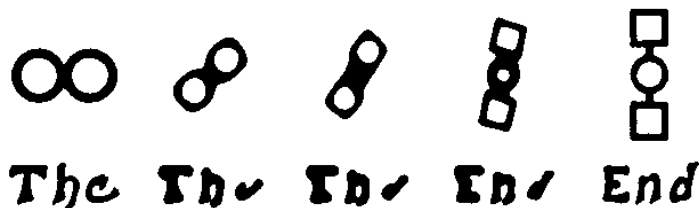


Figure 5: Two shape transformation sequences (using the variational implicit technique) that incorporate warping.

an orthogonal issue to those of implicit function creation and interpolation. Although it is not a focus of our research, for completeness we briefly describe warping in the appendix.

Why has this implicit function building method not been tried using other ways of creating implicit functions? Why not, for example, build a signed distance function in one higher dimension? Because a *complete* description of an object's boundary is required in order to build a signed distance function. When we embed our two shapes into a higher dimension, we only know a *piece* of the boundary of our desired higher-dimensional shape, namely the cross-sections that match the two given objects. In contrast, a complete boundary representation is *not* required when using variational interpolation to create an implicit function. Variational interpolation creates plausible function values in regions where we have no information, and especially in the “unknown” region between the two planes that contain all of our constraints. This plausibility of values comes from the smooth nature of the functions that are created by the variational approach.

## 5.2 Three-Dimensional Shape Transformation

Just as we create a 3D function to create a transformation between 2D shapes, we can move to 4D in order to create a sequence between 3D shapes. We perform shape interpolation between two 3D objects using boundary and normal constraints for each shape. We place the constraints from two 3D objects into four dimensional space, just as we placed constraints from 2D contours into 3D. Similar to contour interpolation, the constraints are separated from one another in the fourth dimension by some specified distance. We place all the constraints from the first object at  $t = 0$ , and the constraints from the second object are placed at  $t = t_{max}$ , where  $t_{max}$  is the given separation distance. We then create a 4D implicit function using variational interpolation. An intermediate shape between the two given shapes is found by extracting the isosurface of a 3D “slice” (actually a volume) of the resulting 4D function.

Figure 6 shows two 3D shape transformation sequence that were constructed using this method. To extract these surfaces we use code published by Bloomenthal that begins at a seed location on the surface of a model and only evaluates the implicit function at points near previously visited locations [4]. This is far more efficient than sampling an entire volume of the implicit function and then extracting an isosurface from the volume. The matrix solution for the transformation sequence of Figure 6 (left) required 13.5 minutes, and each isosurface in the sequence took approximately 2.3 minutes to generate on an SGI Indigo2 with a 195 MHz R10000 processor. Figure 6 (right) shows a transformation between 3D shapes that used warping to align features.

## 6 Surface Reconstruction from Contours

So far we have only considered shape transformation between pairs of objects. In medical reconstruction, however, it is often necessary to create a surface from a large number of parallel 2D slices. Can't we just perform shape interpolation between pairs of slices and stack the results together to create one surface in 3D? Although this method will create a continuous surface, it is almost certain to produce a shape that has surface normal discontinuities at the planes of the original slices. In the plane of slice  $i$ , the surface created between slice pairs  $i - 1$  and  $i$  will usually not agree in surface normal with the surface created between slices  $i$  and  $i + 1$ . Nearly all contour interpolation methods consider only pairs of contours at any one time, and thus suffer from such discontinuities. (A notable exception is [1]).

To avoid discontinuities in surface normal, we must use information about more than just two slices at a given time. We can accomplish this using a generalization of the variational approach to shape transformation. Assume that we begin with  $k$  sets of constraints, one set for each 2D data slice. Instead of considering the contours in pairs, we place the constraints for all of the  $k$  slices into

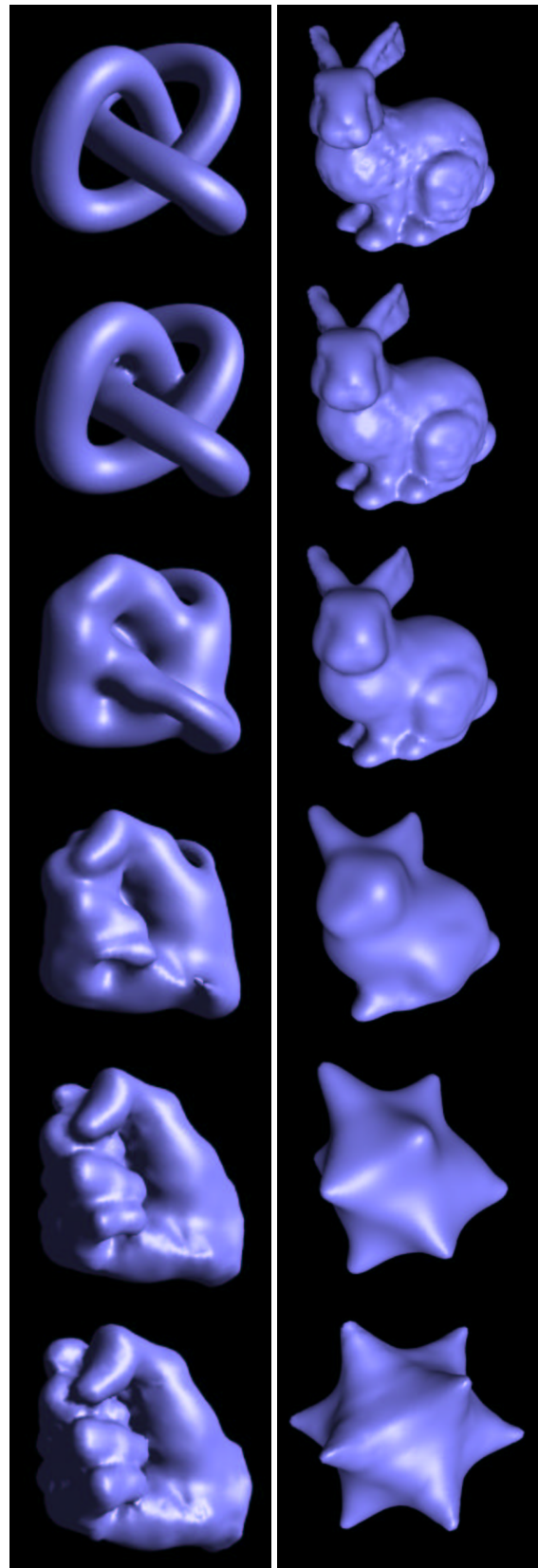


Figure 6: 3D shape transformation sequences.

3D simultaneously. Specifically, the constraints of slice  $i$  are placed in the plane  $z = si$ , where  $s$  is the spacing between planes. Once the constraints from *all* slices have been placed in 3D, we invoke

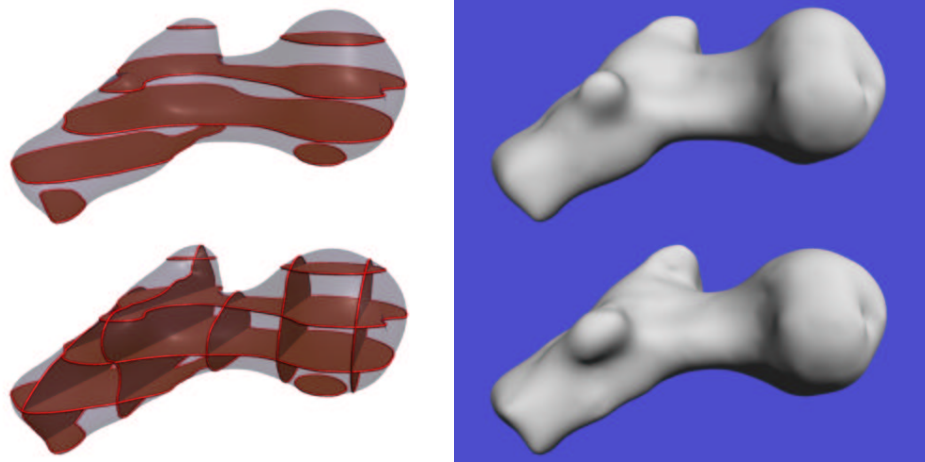


Figure 7: Reconstruction of hip joint from contours. Top row shows the five parallel slices used and the final surface. Bottom row shows intersecting contours and the more detailed surface that is created.

variational interpolation *once* to create a single implicit function in 3D. The zero-valued isosurface exactly passes through each contour of the data. Due to the smooth nature of variational interpolation, the gradient of the implicit function is everywhere continuous. This means that surface normal discontinuities are rare, appearing in pathological situations when the gradient vanishes such as when two features just barely touch. Figure 7 (top row) illustrates the result of this contour interpolation approach. The hip joint reconstruction in the upper right was created from the five slices shown at the upper left.

A side benefit of using the variational implicit function method is that it produces smoothly rounded caps on the ends of surfaces. Notice that in Figure 7 (top left) that the reconstructed surface extends beyond the constraints in the positive and negative  $z$  direction (the direction of slice stacking). This “rounding off” of the ends is a natural side effect of variational interpolation, and need not be explicitly specified.

### 6.1 Non-Parallel Contours

In the previous section, we only considered placing constraints within planes that are all parallel to one another. There is nothing special about any particular set of planes, however, once we are specifying constraints in 3D. We can mix together constraints that are taken from planes at any angle whatsoever, so long as we know the relative positions of the planes (and thus the constraints). Most contour interpolation procedures cannot integrate data taken from slices in several directions, but the variational approach allows complete freedom in this regard. Figure 7 (lower row) shows several contours that are placed perpendicular to one another, and the result of using variational interpolation on the group of constraints from these contours.

### 6.2 Between-Contour Spacing

Up to this point we have not discussed the separating distance  $s$  between the slices that contain the contour data. This separating distance has a concrete meaning in medical shape reconstruction from 2D contours. Here we know the actual 3D separation between the contours from the data capture process. This “natural” distance is the separating distance  $s$  that should be used when reconstructing the surface using variational interpolation. Upon reflection, it is odd that some contour interpolation methods do not make use of the data capture distance between slices. In some cases a medical technician will deliberately vary the spacing between data slices in order to capture more data in a particular region of interest. Using variational interpolation, we may incorporate this information

about varying separation distances into the surface reconstruction process.

For both special effects production and for computer aided design, the distance between the separating planes can be thought of as a control knob for the artist or designer. If the distance is small, only pairs of features from the two shapes that are very close to one another will be preserved through all the intermediate shapes. If the separation distance is large, the intermediate shape is guided by more global properties of the two shapes. In some sense, the separating distance specifies whether the shape transformation is local or global in nature. The separation distance is just one control knob for the user, and in the next section we will explore another user control.

## 7 Influence Shapes

In this section we present a method of controlling shape transformation by introducing an *influence shape*. The idea to use additional objects as controls for shape transformation was introduced by Rossignac and Kaul [24]. Such intermediate shape control can be performed in a natural way using variational interpolation. The key is to step into a still higher dimension when performing shape transformation.

Recall that to create a transformation sequence between two given shapes we added one new dimension, called  $t$  earlier. We can think of the two shapes as being two points that are separated along the  $t$  dimension, and these two points are connected by a line segment that joins the two points along this dimension. If we begin with three shapes, however, we can in effect place them at the three points of a triangle. In order to do so we need not just one additional dimension but two, call them  $s$  and  $t$ .

As an example, we may begin with three different 3D shapes A, B and C. To each constraint that describes one of the shapes, we can add two new coordinates,  $s$  and  $t$ . Constraints from shape A at  $(x, y, z)$  are placed at  $(x, y, z, 0, 0)$ , constraints from shape B are placed at  $(x, y, z, 1, 0)$  and shape C constraints are placed at  $(x, y, z, 1/2, 1/2)$ . Variational interpolation based on these 5-dimensional constraints results in a 5D implicit function. Three-dimensional slices of this function along the  $s$ -dimension between 0 and 1 are simply shape sequences between shapes A and B when the  $t$ -dimension value is fixed at zero. If, however, the  $t$ -dimension value is allowed to become positive as  $s$  varies from 0 to 1, then the intermediate shapes will take on some of the characteristics of shape C. In fact, the 5D implicit function actually captures an entire family of shapes that are various blends between the three shapes. Figure 8 illustrates some members of such a family of shapes.



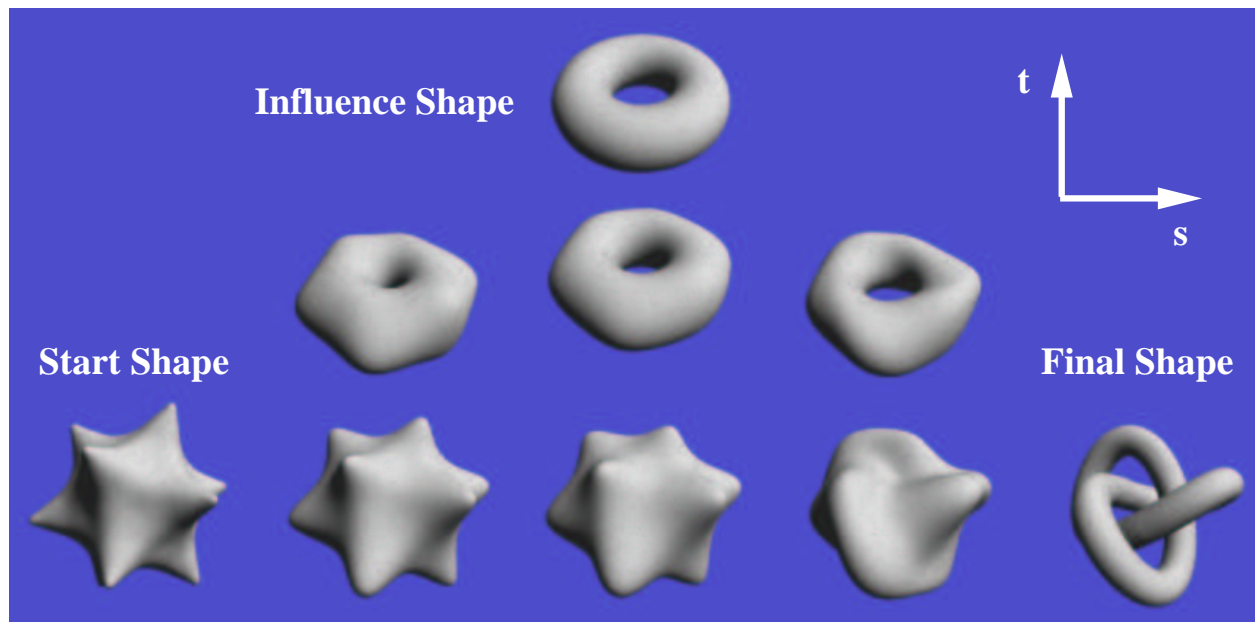


Figure 8: Sequence between star and knot can be influenced by a torus (the influence shape) if the path passes near the torus in the five-dimensional space.

There is no reason to stop at three shapes. It is possible to place four shapes at the corners of a quadrilateral, five shapes around a pentagon, and so on. If we wish to use four shapes, then placing the constraints at the corners of a quadrilateral using two additional dimensions would not allow us to produce a shape that was arbitrary mixtures between the shapes. In order to do so, we can place the constraints in yet a higher dimension, in effect placing the four shapes at the corners of a tetrahedron in  $N + 3$  dimensions, where  $N$  is the dimension of the given shapes.

There are two related themes that guide our technique for shape transformation. The first is that shape transformation should be thought of as a shape-creation problem in a higher dimension. The second theme is that better shape transformation sequences are produced when all of the problem constraints are solved simultaneously—in our case by using variational interpolation. Influence shapes are the result of taking these ideas to an extreme.

## 8 Conclusions and Future Work

Our new approach uses variational interpolation to produce one implicit function that describes an entire sequence of shapes. Specific characteristics of this approach include:

- Smooth intermediate shapes
- Shape transformation in any number of dimensions
- Analytic solutions that are free of polygon and voxel artifacts
- Continuous surface normals for contour interpolation
- Contour slices may be at any orientation, even intersecting

This approach provides two new controls for creating shape transformation sequences:

- Separation distance gives local/global interpolation tradeoff
- May use influence shapes to control a transformation

The approach we have presented in this paper re-formulates the shape interpolation problem as an interpolation problem in one higher dimension. In essence, we treat the “time” dimension just like another spatial dimension. We have found that using the variational interpolation method produces excellent results, but the mathematical literature abounds with other interpolation methods. An exciting avenue for future work is to investigate what other interpolation techniques can also be used to create implicit functions for shape transformation. Another issue is whether shape transformation methods can be made fast enough to allow a user interactive control. Finally, how might surface properties such as color and texture be carried through intermediate objects?

## 9 Acknowledgements

This work owes a good deal to Andrew Glassner for getting us interested in the shape transformation problem. We thank our colleagues and the anonymous reviewers for their helpful suggestions. This work was funded by ONR grant N00014-97-1-0223.

## References

- [1] Barequet, Gill, Daniel Shapiro and Ayellet Tal, “History Consideration in Reconstructing Polyhedral Surfaces from Parallel Slices,” *Proceedings of Visualization '96*, San Francisco, California, Oct. 27 – Nov. 1, 1996, pp. 149–156.
- [2] Barr, Alan H., “Global and Local Deformations of Solid Primitives,” *Computer Graphics*, Vol. 18, No. 3 (SIGGRAPH 84), pp. 21–30.
- [3] Beier, Thaddeus and Shawn Neely, “Feature-Based Image Metamorphosis,” *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 35–42.
- [4] Bloomenthal, Jules, “An Implicit Surface Polygonizer,” in *Graphics Gems IV*, edited by Paul S. Heckbert, Academic Press, 1994, pp. 324–349.
- [5] Bookstein, Fred L., “Principal Warps: Thin Plate Splines and the Decomposition of Deformations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 6, June 1989, pp. 567–585.
- [6] Celniker, George and Dave Gossard, “Deformable Curve and Surface Finite-Elements for Free-Form Shape Design,” *Computer Graphics*, Vol. 25, No. 4 (SIGGRAPH 91), July 1991, pp. 257–266.

- [7] Cohen-Or, Daniel, David Levin and Amira Solomovici, “Three Dimensional Distance Field Metamorphosis,” *ACM Transactions on Graphics*, 1997.
- [8] Duchon, Jean, “Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces,” in *Constructive Theory of Functions of Several Variables*, Lecture Notes in Mathematics, edited by A. Dolb and B. Eckmann, Springer-Verlag, 1977, pp. 85–100.
- [9] Duncan, Jody, “A Once and Future War,” *Cinefex*, No. 47 (entire issue devoted to the film Terminator 2), August 1991, pp. 4–59.
- [10] Fuchs, H., Z. M. Kedem and S. P. Uelson, “Optimal Surface Reconstruction from Planar Contours,” *Communications of the ACM*, Vol. 20, No. 10, October 1977, pp. 693–702.
- [11] Golub, Gene H. and Charles F. Ban Loan, *Matrix Computations*, John Hopkins University Press, 1996.
- [12] Gregory, Arthur, Andrei State, Ming C. Lin, Dinesh Manocha, Mark A. Livingston, “Feature-based Surface Decomposition for Correspondence and Morphing between Polyhedra,” *Proceedings of Computer Animation*, Philadelphia, PA., 1998.
- [13] He, Taosong, Sidney Wang and Arie Kaufman, “Wavelet- Based Volume Morphing,” *Proceedings of Visualization '94*, Washington, D. C., edited by Daniel Bergeron and Arie Kaufman, October 17–21, 1994, pp. 85–92.
- [14] Herman, Gabor T., Jingsheng Zheng and Carolyn A. Bucholtz, “Shape-Based Interpolation,” *IEEE Computer Graphics and Applications*, Vol. 12, No. 3 (May 1992), pp. 69–79.
- [15] Hughes, John F., “Scheduled Fourier Volume Morphing,” *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 43–46.
- [16] Kaul, Anil and Jarek Rossignac, “Solid- Interpolating Deformations: Construction and animation of PIPs,” *Proceedings of Eurographics '91*, Vienna, Austria, 2–6 Sept. 1991, pp. 493–505.
- [17] Kent, James R., Wayne E. Carlson and Richard E. Parent, “Shape Transformation for Polyhedral Objects,” *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 47–54.
- [18] Leros, Apostolos, Chase Garfinkle and Marc Levoy, “Feature-Based Volume Metamorphosis,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 95), pp. 449–456.
- [19] Levin, David, “Multidimensional Reconstruction by Set-valued Approximation,” *IMA J. Numerical Analysis*, Vol. 6, 1986, pp. 173–184.
- [20] Litwinowicz, Peter and Lance Williams, “Animating Images with Drawings,” *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 94), pp. 409–412.
- [21] Lorensen, William and Harvey E. Cline, “Marching Cubes: A High Resolution 3-D Surface Construction Algorithm,” *Computer Graphics*, Vol. 21, No. 4 (SIGGRAPH 87), July 1987, pp. 163–169.
- [22] Meyers, David and Shelley Skinner, “Surfaces From Contours: The Correspondence and Branching Problems,” *Proceedings of Graphics Interface '91*, Calgary, Alberta, 3–7 June 1991, pp. 246–254.
- [23] Payne, Bradley A. and Arthur W. Toga, “Distance Field Manipulation of Surface Models,” *IEEE Computer Graphics and Applications*, Vol. 12, No. 1, January 1992, pp. 65–71.
- [24] Rossignac, Jarek and Anil Kaul, “AGRELS and BIPs: Metamorphosis as a Bezier Curve in the Space of Polyhedra,” *Proceedings of Eurographics '94*, Oslo, Norway, Sept. 12–16, 1994, pp. 179–184.
- [25] Sederberg, Thomas W. and Eugene Greenwood, “A Physically Based Approach to 2-D Shape Blending,” *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 25–34.
- [26] Sederberg, Thomas W. and Scott R. Parry, “Free-Form Deformations of Solid Geometric Models,” *Computer Graphics*, Vol. 20, No. 4 (SIGGRAPH 86), pp. 151–160.
- [27] Turk, Greg and James F. O’Brien, “Variational Implicit Surfaces,” Tech Report GIT-GVU-99-15, Georgia Institute of Technology, May 1999, 9 pages.
- [28] Wolberg, George, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, California 1990.

## 10 Appendix: Warping

Warping is a commonly used method of providing user control of shape interpolation. Although warping is not a focus of our research, for the sake of completeness we describe below how warping may be used together with our shape transformation technique. Research on warping (sometimes called deformation) include [2, 26, 28, 3, 18, 7].

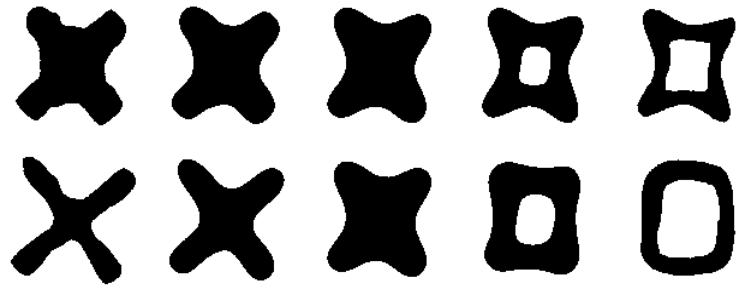


Figure 9: The extreme left and right shapes in the top row have been warped before creating the upper shape transformation sequence. The lower row is an un-warped version of this sequence that gives the final transformation from an X to O.

For symmetry, we choose to warp each shape “half-way” to the other shape. Given a set of user-supplied corresponding points between two shapes  $A$  and  $B$ , we construct two displacement warp functions  $w_A$  and  $w_B$ . The function  $w_A$  specifies what values to add to locations on shape  $A$  in order to warp it half-way to shape  $B$ , and the warping function  $w_B$  warps  $B$  half of the way to  $A$ .

In what follows, we will describe the warping process for two-dimensional shapes. Let  $\{a_1, a_2, \dots, a_k\}$  be a set of points on shape  $A$ , and let  $\{b_1, b_2, \dots, b_k\}$  be the corresponding points on  $B$ . We construct the two functions  $w_A$  and  $w_B$  such that  $w_A(a_i) = (b_i - a_i)/2$  and  $w_B(b_i) = (a_i - b_i)/2$  hold for all  $i$ . Constructing these functions is another example of scattered data interpolation which we can solve using variational techniques. For 2D shapes, if  $a_i = (a_i^x, a_i^y)$  and  $b_i = (b_i^x, b_i^y)$ , then the  $x$  component  $w_A^x$  of the displacement warp  $w_A$  has  $k$  constraints at the positions  $a_i$  with values  $(b_i^x - a_i^x)/2$ . We invoke variational interpolation to satisfy these constraints, and do the same to construct the  $y$  component of the warp. The function  $w_B$  is constructed similarly. This is not a new technique, and researchers who use thin-plate techniques to perform shape warping include [5, 20] and others.

In order to combine warping with shape transformation, we use these functions to displace all of the boundary constraints of the given shapes. These displaced boundary constraints are embedded in 3D (as described in Section 5) and then variational interpolation is used to create the implicit function that describes the entire shape transformation sequence. The result of this process is a three-dimensional implicit function, each slice of which is an intermediate shape between two warped shapes. The top row of Figure 9 shows such warped intermediate shapes. We can think of the two “ends” of this implicit function (at  $t = 0$  and  $t = t_{max}$ ) as being warped versions of our original shapes. In order to match the two original shapes, the surface of this 3D implicit function needs to be unwrapped. To simplify the equations, assume that the value of  $t_{max}$  is 2. If  $t \leq 1$  the unwarping function  $u(x, y, t)$  is:

$$u(x, y, t) = (x + (1 - t)w_A^x(x, y), y + (1 - t)w_A^y(x, y), t) \quad (4)$$

If  $t > 1$  then the unwarping function is:

$$u(x, y, t) = (x + (t - 1)w_B^x(x, y), y + (t - 1)w_B^y(x, y), t) \quad (5)$$

At the extreme of  $t = 0$ , the warp  $u(x, y, t)$  un-does the warping we used for the first shape. At  $t = 2$ , the function  $u(x, y, t)$  reverses the warping used for the second shape. When  $t = 1$  (the middle shape in the sequence), no warp is performed. The bottom sequence of shapes in Figure 9 shows the result of the entire shape transformation process that includes warping. Both sequences in Figure 9 were created using warping in addition to shape transformation.

Although we have described the warping process for 2D shapes, the same method may be used for shape transformation between 3D shapes. For Figure 6 (right), warping was used to align the bunny ears to the points of the star.

# Modelling with Implicit Surfaces that Interpolate

GREG TURK

Georgia Institute of Technology

JAMES F. O'BRIEN

University of California, Berkeley

---

We introduce new techniques for modelling with *interpolating implicit surfaces*. This form of implicit surface was first used for problems of surface reconstruction and shape transformation, but the emphasis of our work is on model creation. These implicit surfaces are described by specifying locations in 3D through which the surface should pass, and also identifying locations that are interior or exterior to the surface. A 3D implicit function is created from these constraints using a variational scattered data interpolation approach, and the iso-surface of this function describes a surface. Like other implicit surface descriptions, these surfaces can be used for CSG and interference detection, may be interactively manipulated, are readily approximated by polygonal tilings, and are easy to ray trace. A key strength for model creation is that interpolating implicit surfaces allow the direct specification of both the location of points on the surface and the surface normals. These are two important manipulation techniques that are difficult to achieve using other implicit surface representations such as sums of spherical or ellipsoidal Gaussian functions ("blobbies"). We show that these properties make this form of implicit surface particularly attractive for interactive sculpting using the particle sampling technique introduced by Witkin and Heckbert. Our formulation also yields a simple method for converting a polygonal model to a smooth implicit model, as well as a new way to form blends between objects.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms: Algorithms

Additional Key Words and Phrases: Implicit surfaces, thin-plate techniques, function interpolation, modeling

---

## 1. INTRODUCTION

The computer graphics, computer-aided design and computer vision literatures are filled with an amazingly diverse array of approaches to surface description. The reason for this variety is that there is no single representation of surfaces that satisfies the needs of every problem in every application area. This paper is about modelling with *interpolating implicit surfaces*, a surface representation that we believe will be useful in several areas in 3D modeling. These implicit surfaces are smooth, exactly pass through a set of given constraint points, and can describe closed surfaces of arbitrary topology.

In order to illustrate our basic approach, Figure 1 (left) shows an interpolating implicit curve, the 2D analog of an interpolating implicit surface. The small open circles in this figure indicate the location of constraints where the 2D implicit function must take on the value of zero. The single plus sign corresponds to an additional constraint where the implicit function must take on the value of some

---

This work was funded under ONR grant N00014-97-0223.

Authors' addresses: G. Turk, College of Computing, Room 257, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280; email: turk@cc.gatech.edu; J. F. O'Brien, Soda Hall, Mail Code 1776, EECS Computer Science Division, University of California, Berkeley, Berkeley, CA 94720-1776; email: job@eecs.berkeley.edu.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2002 ACM 0730-0301/02/1000-0855 \$5.00

ACM Transactions on Graphics, Vol. 21, No. 4, October 2002, Pages 855–873.

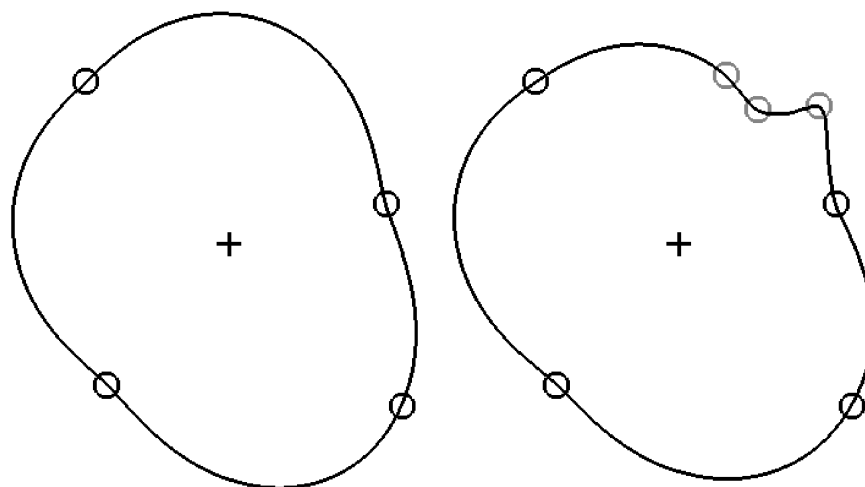


Fig. 1. Curves defined using interpolating implicit functions. The curve on the left is defined by four zero-valued and one positive, constraint. This curve is refined by adding three new zero-valued constraints (shown in red at right).

arbitrary positive constant, which for this example is one. These constraints are passed along to a scattered data interpolation routine that generates a smooth 2D function meeting the given constraints. The desired curve is defined to be the locus of points at which the function takes on the value of zero. The curve exactly passes through each of the zero-value constraints, and its defining function is positive inside this curve and negative outside. For this 2D example, we use a variational technique that minimizes the aggregate curvature of the function that it creates, and this technique for creating a function is often referred to as thin-plate interpolation.

We can create surfaces in 3D in exactly the same way as the 2D curves in Figure 1. Zero-valued constraints are defined by the modeler at 3D locations, and positive values are specified at one or more places that are to be interior to the surface. A variational interpolation technique is then invoked that creates a scalar-valued function over a 3D domain. The desired surface is simply the set of all points at which this scalar function takes on the value zero. Figure 2 (left) shows a surface that was created in this fashion by placing four zero-valued constraints at the vertices of a regular tetrahedron and placing a single positive constraint in the center of the tetrahedron. The result is a nearly spherical surface. More complex surfaces such as the branching shape in Figure 2 (right) can be defined simply by specifying more constraints. Figure 3 shows an example of an interpolating implicit surface created from polygonal data.

The remainder of this paper is organized as follows. In Section 2 we examine related work, including implicit surfaces and thin-plate interpolation techniques. We describe in Section 3 the mathematical framework for solving variational problems using radial basis functions. Section 4 presents three strategies that may be used together with variational methods to create implicit surfaces. These strategies differ in where they place the non-zero constraints. In Section 5 we show that interpolating implicit surfaces are well suited for interactive sculpting. In Section 6 we present a new method of creating soft blends between objects, based on interpolating implicit functions. Section 7 describes two rendering techniques, one that relies on polygonal tiling and another based on ray tracing. In Section 8 we compare interpolating implicit surfaces with traditional thin-plate surface modeling and with implicit functions that are created using ellipsoidal Gaussian functions. Finally, Section 9 indicates potential applications and directions for future research.

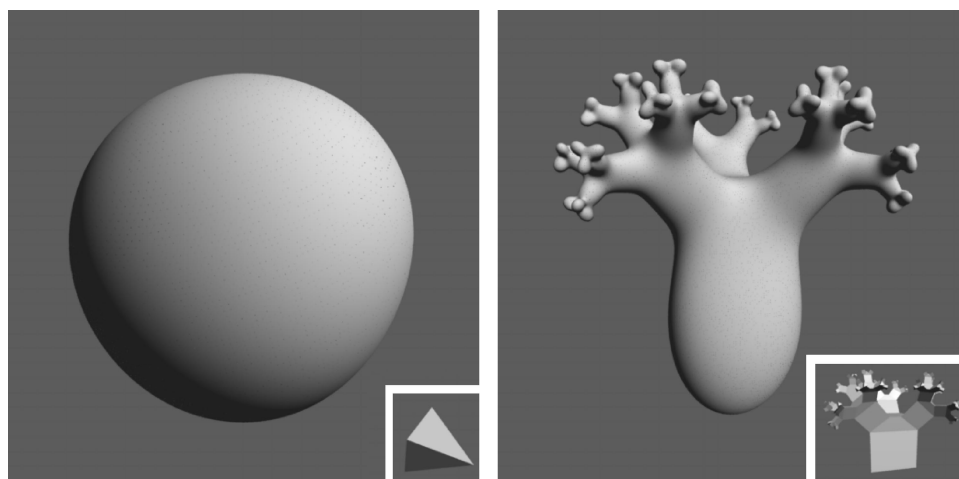


Fig. 2. Surfaces defined by interpolating implicit functions. The left surface is defined by zero-valued constraints at the corners of a tetrahedron and one positive constraint in its center. The branching surface at the right was created using constraints from the vertices of the inset polygonal object.

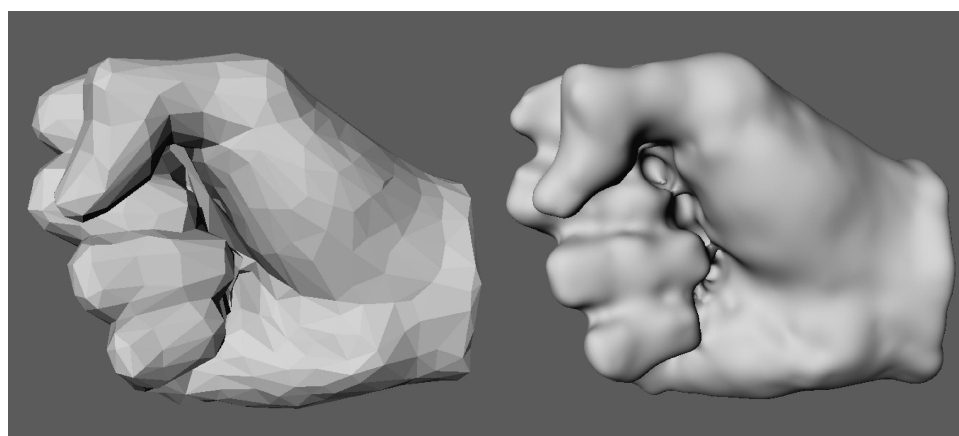


Fig. 3. Polygonal surface of a human fist with 750 vertices (left) and an interpolating implicit surface created from the polygons (right).

## 2. BACKGROUND AND RELATED WORK

Interpolating implicit surfaces draw upon two areas of modeling: implicit surfaces and thin-plate interpolation. In this section we briefly review work in these two sub-areas. Interpolating implicit surfaces are not new to graphics, and at the close of this section we describe earlier published methods of creating interpolating implicit surfaces.

### 2.1 Implicit Surfaces

An implicit surface is defined by an implicit function, a continuous scalar-valued function over the domain  $\mathbf{R}^3$ . The implicit surface of such a function is the locus of points at which the function takes on the value zero. For example, a unit sphere may be defined using the implicit function  $f(\mathbf{x}) = 1 - |\mathbf{x}|$ , for

ACM Transactions on Graphics, Vol. 21, No. 4, October 2002.

points  $\mathbf{x} \in \mathbf{R}^3$ . Points on the sphere are those locations at which  $f(\mathbf{x}) = 0$ . This implicit function takes on positive values inside the sphere and is negative outside the surface, as will be the convention in this paper.

An important class of implicit surfaces are the *blobby* or *meta-ball* surfaces [Blinn 1982; Nishimura et al. 1985]. The implicit functions of these surfaces are the sum of radially symmetric functions that have a Gaussian profile. Here is the general form of such an implicit function:

$$f(\mathbf{x}) = -t + \sum_{i=1}^n g_i(\mathbf{x}) \quad (1)$$

In the above equation, a single function  $g_i$  describes the profile of a “blobby sphere” (a Gaussian function) that has a particular center and standard deviation. The bold letter  $\mathbf{x}$  represents a point in the domain of our implicit function, and in this paper we will use bold letters to represent such points, both in 2D and 3D. The value  $t$  is the iso-surface threshold, and it specifies one particular surface from a family of nested surfaces that are defined by the sum of Gaussians. When the centers of two blobby spheres are close enough to one another, the implicit surface appears as though the two spheres have melted together. A typical form for a blobby sphere function  $g_i$  is the following:

$$g_i(\mathbf{x}) = e^{|\mathbf{x} - \mathbf{c}_i|^2 / \sigma_i^2} \quad (2)$$

In this equation, the constant  $\sigma_i$  specifies the standard deviation of the Gaussian function, and thus is the control over the radius of a blobby sphere. The center of a blobby sphere is given by  $\mathbf{c}_i$ . Evaluating an exponential function is computationally expensive, so some authors have used piecewise polynomial expressions instead of exponentials to define these blobby sphere functions [Nishimura et al. 1985; Wyvill et al. 1986]. A greater variety of shapes can be created with the blobby approach by using ellipsoidal rather than spherical functions.

Another important class of implicit surfaces are the algebraic surfaces. These are surfaces that are described by polynomial expressions in  $x$ ,  $y$  and  $z$ . If a surface is simple enough, it may be described by a single polynomial expression. A good deal of attention has been devoted to this approach, and we recommend Taubin [1993] and Keren and Gotsman [1998] as starting points in this area. Much of the work on this method has been devoted to fitting an algebraic surface to a given collection of points. Usually it is not possible to interpolate all of the data points, so error minimizing techniques are sought. Surfaces may also be described by piecing together many separate algebraic surface patches, and here again there is a large literature on the subject. Good introductions to these surfaces may be found in the chapters by Bajaj and Rockwood in Bloomenthal [1997]. It is easier to create complex surfaces using a collection of algebraic patches rather than using a single algebraic surface. The tradeoff, however, is that a good deal of machinery is required to create smooth joins across patch boundaries.

We have only described some of the implicit surface representations that are most closely related to our own work. There are many other topics within the broad area of implicit surfaces, and we refer the interested reader to the excellent book by Bloomenthal and his co-authors, Bloomenthal [1997].

## 2.2 Thin-Plate Interpolation

Thin-plate spline surfaces are a class of height fields that are closely related to the interpolating implicit surfaces of this paper. Thin-plate interpolation is one approach to solving the *scattered data interpolation* problem. The two-dimensional version of this problem can be stated as follows: Given a collection of  $k$  constraint points  $\{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k\}$  that are scattered in the  $xy$ -plane, together with scalar height values at each of these points  $\{h_1, h_2, \dots, h_k\}$ , construct a “smooth” surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function  $f(\mathbf{x})$  so that

$f(\mathbf{c}_i) = h_i$ , for  $1 \leq i \leq k$ . If we define the word *smooth* in a particular way, there is a unique solution to such a problem, and this solution is the thin-plate interpolation of the points. Consider the energy function  $E(f)$  that measures the smoothness of a function  $f$ :

$$E(f) = \int_{\Omega} f_{xx}^2(\mathbf{x}) + 2f_{xy}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x}) d\mathbf{x} \quad (3)$$

The notation  $f_{xx}$  means the second partial derivative in the  $x$  direction, and the other two terms are similar partial derivatives, one of them mixed. This energy function is basically a measure of the aggregate curvature of  $f(\mathbf{x})$  over the region of interest  $\Omega$  (a portion of the plane). Any creases or pinches in a surface will result in a larger value of  $E$ . A smooth function that has no such regions of high curvature will have a lower value of  $E$ . Note that because there are only squared terms in the integral, the value for  $E$  can never be negative. The thin-plate solution to an interpolation problem is the function  $f(\mathbf{x})$  that satisfies all of the constraints and that has the smallest possible value of  $E$ . Note that thin-plate surfaces are height fields, and thus they are in fact *parametric* surfaces.

This interpolation method gets its name because it is much like taking a thin sheet of metal, laying it horizontally and bending it so that it just touches the tips of a collection of vertical poles that are set at the positions and heights given by the constraints of the interpolation problem. The metal plate resists bending so that it smoothly changes its height in the positions between the poles. This springy resistance is mimicked by the energy function  $E$ . Thin-plate interpolation is often used in the computer vision domain, where there are often sparse surface constraints [Grimson 1983; Terzopoulos 1988]. The above curvature minimization process is sometimes referred to as regularization, and can be thought of as an additional constraint that selects a unique surface out of an infinite number of surfaces that match a set of given height constraints. Solving such constrained problems draws from a branch of mathematics called the variational calculus, thus thin-plate techniques are sometimes referred to as variational methods.

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points  $\mathbf{c}_i$  are positions in  $n$ -dimensions rather than in 2D, this is called the  $n$ -dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for any dimension. In this paper we will make use of variational interpolation in two and three dimensions.

### 2.3 Related Work on Implicit Surfaces

The first publication on interpolating implicit surfaces of which we are aware is that of Savchenko et al. [1995]. We consider this to be a pioneering paper in implicit surfaces, and feel it deserves to be known more widely than it is at present. Their research was on the creation of implicit surfaces from measured data such as range data or contours. Their work did not, however, describe techniques for modelling. Their approach to implicit function creation is similar to our method in the present paper in that both solve a linear system to get the weights for radial basis functions. The work of Savchenko et al. differs from our own in that they use a *carrier solid* to suggest what part of space should be interior to the surface that is being created. We believe that the three methods that we describe for defining the interior of a surface in Section 4 of this paper give more user control than a carrier solid and are thus more appropriate for modelling.

The implicit surface creation methods described in this paper are an outgrowth of earlier work in shape transformation by Turk and Turk and O'Brien [1999]. They created implicit functions in  $n + 1$  dimensions to interpolate between pairs of  $n$ -dimensional shapes. These implicit functions were created using the *normal constraint* formulation of interpolating implicit surfaces, as described in Section 4.3 of

this paper. The present paper differs from that of Turk and O'Brien [1999] in its introduction of several techniques for defining interpolating implicit surfaces that are especially useful for model creation.

Recently, techniques have developed that allow the methods discussed above to be applied to systems with large numbers of constraints [Morse et al. 2001; Carr et al. 2001]. The work of Morse et al. uses Gaussian-like compactly supported radial basis functions to accelerate the surface building process; they are able to create surfaces that have tens of thousands of constraints. Carr et al. use fast evaluation methods to reconstruct surfaces using up to a half millions basis functions. They use the radial basis function  $\phi(\mathbf{x}) = |\mathbf{x}|$ , the biharmonic basis function. Both of these improvements for creating surfaces with many constraints are complementary to the work of the present paper, and the new techniques that we describe in Sections 4, 5 and 6 should work gracefully with the methods in both of these papers.

### 3. VARIATIONAL METHODS AND RADIAL BASIS FUNCTIONS

In this section we review the necessary mathematical background for thin-plate interpolation. This will provide the tools that we will then use in Section 4 to create interpolating implicit surfaces.

The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function,  $f(\mathbf{x})$ , that will minimize Equation 3 subject to the interpolation constraints  $f(\mathbf{c}_i) = h_i$ . There are several numerical methods that can be used to solve this type of problem. Two commonly used methods, finite elements and finite differencing techniques, discretize the region of interest,  $\Omega$ , into a set of cells or elements and define local basis functions over the elements. The function  $f(\mathbf{x})$  can then be expressed as a linear combination of the basis functions so that a solution can be found, or approximated, by determining suitable weights for each of the basis functions. This approach has been widely used for height-field interpolation and deformable models, and examples of its use can be found in [Terzopoulos 1988; Szeliski 1990; Celniker and Gossard 1991; Welch and Witkin 1994]. While finite elements and finite differencing techniques have proven useful for many problems, the fact that they rely on discretization of the function's domain is not always ideal. Problems that can arise due to discretization include visibly stair-stepped surfaces and the inability to represent fine details. In addition, the cost of using such methods grows cubically with the desired resolution.

An alternate approach is to express the solution in terms of radial basis functions centered at the constraint locations. Radial basis functions are radially symmetric about a single point, or center, and they have been widely used for function approximation. Remarkably, it is possible to choose these radial functions in such a way that they will automatically solve differential equations, such as the one required to solve Equation 3, subject to constraints located at their centers. For the 2D interpolation problem, Equation 3 can be solved using the biharmonic radial basis function:

$$\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|) \quad (4)$$

This is commonly known as the thin-plate radial basis function. For 3D interpolation, one commonly used radial basis function is  $\phi(\mathbf{x}) = |\mathbf{x}|^3$ , and this is the basis function that we use. We note that Carr et al. [2001] used the basis function  $\phi(\mathbf{x}) = |\mathbf{x}|$ . Duchon [1977] did much of the early work on variational interpolation, and the report by Girosi, Jones and Poggio is a good entry point into the mathematics of variational interpolation [Girosi et al. 1993].

Using the appropriate radial basis functions, we can write the interpolation function in this form:

$$f(\mathbf{x}) = \sum_{j=1}^k w_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \quad (5)$$



In the above equation,  $\mathbf{c}_j$  are the locations of the constraints,  $w_j$  are the weights, and  $P(\mathbf{x})$  is a degree one polynomial that accounts for the linear and constant portions of  $f$ . Solving for the weights  $w_j$  and the coefficients of  $P(\mathbf{x})$  subject to the given constraints yields a function that both interpolates the constraints and minimizes Equation 3. The resulting function exactly interpolates the constraints (if we ignore numerical precision issues), and is not subject to approximation or discretization errors. Also, the number of weights to be determined does not grow with the size of the region of interest  $\Omega$ . Rather, it is only dependent on the number of constraints.

To solve for the set of  $w_j$  that will satisfy the interpolation constraints, we begin with the criteria that the surface must interpolate our constraints:

$$h_i = f(\mathbf{c}_i) \quad (6)$$

We now substitute the right side of Equation 5 for  $f(\mathbf{c}_i)$  to give us:

$$h_i = \sum_{j=1}^k w_j \phi(\mathbf{c}_i - \mathbf{c}_j) + P(\mathbf{c}_i) \quad (7)$$

Since the above equation is linear with respect to the unknowns,  $w_j$ , and the coefficients of  $P(\mathbf{x})$ , it can be formulated as a linear system. For interpolation in 3D, let  $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$  and let  $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$ . Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \dots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (8)$$

The sub-matrix in Equation 8 consisting of the  $\phi_{ij}$ 's is conditionally positive-definite on the subspace of vectors that are orthogonal to the last four rows of the full matrix, so Equation 8 is guaranteed to have a solution. We used symmetric LU decomposition to solve this system of equations for all of the examples shown in this paper. Our implementation to set up the system, call the LU decomposition routine and evaluate the interpolating function of Equation 5 for a given  $\mathbf{x}$  consists of about 100 lines of commented C++ code. This code plus the public-domain polygonalization routine described in Section 7.1 is all that is needed to create interpolating implicit surfaces.

Two concerns that arise with such matrix systems are computation times and ill-conditioned systems. For systems with up to a few thousand centers, including all of the examples in this paper, direct solution techniques such as LU decomposition and SVD are practical. However as the system becomes larger, the amount of work required to solve the system grows as  $O(k^3)$ . We have used direct solution methods for systems with up to roughly 3,000 constraints. LU decomposition becomes impractical for more constraints than this. We are pleased that other researchers, notably the authors of [Morse et al. 2001; Carr et al. 2001], have begun to address this issue of computational complexity.

As the number of constraints grows, the condition number of the matrix in equation 8 is also likely to grow, leading to instability for some solution methods. For the systems we have worked with,

ill-conditioning has not been a problem. If problems arise for larger systems, variational interpolation is such a well-studied problem that methods exist for improving the conditioning of the system of equations (see Dyn [1987]).

#### 4. CREATING INTERPOLATING IMPLICIT SURFACES

With tools for solving the scattered data interpolation problem in hand, we now turn our attention to creating implicit functions. In this section we will examine three ways in which to define an interpolating implicit surface. Common to all three approaches, is the specification of zero-valued constraints through which the surface must pass. The three methods differ in specifying where the implicit function takes on positive and negative values. These methods are based on using three different kinds of constraints: *interior*, *exterior*, and *normal*. We will look at creating both 2D interpolating implicit curves and 3D interpolating implicit surfaces. The 2D curve examples are for illustrative purposes, and our actual goal is the creation of 3D surfaces.

##### 4.1 Interior Constraints

The left portion of Figure 1 (earlier in this paper) shows the first method of describing an interpolating implicit curve. Four zero-valued constraints have been placed in the plane. We call such zero-value constraints *boundary constraints* because these points will be on the boundary between the interior and exterior of the shape that is being defined. In addition to the four boundary constraints, a single constraint with a value of one is placed at the location marked with a plus sign. We use the term *interior constraint* when referring to such a positive-valued constraint that helps to determine the interior of the surface. We construct an implicit function from these five constraints simply by invoking the 2D variational interpolation technique described in earlier sections. The interpolation method returns a set of scalar coefficients  $w_i$  that weight a collection of radially symmetric functions  $\phi$  that are centered at the constraint positions. The implicit curve shown in the figure is given by those locations at which the variationally-defined function takes on the value zero. The function takes on positive values inside the curve and is negative at locations outside the curve. Figure 1 (right) shows a refinement of the curve that is made by adding three more boundary constraints to the original set of constraints in the left portion of the figure.

Why does an interior constraint surrounded by zero-valued constraints yield a function that is negative beyond the boundary constraints? The key is that the energy function is larger for functions that take on positive values on both sides of a zero-valued constraint. Each boundary constraint acts much like a see-saw. If we pull the surface up on one side of a boundary constraint (using an interior constraint), then the other side tends to move down.

Creating surfaces in 3D is accomplished in exactly the same way as the 2D case. Zero-valued constraints are specified by the modeler as those 3D points through which the surfaces should pass, and positive values are specified at one or more places that are to be interior to the surface. Variational interpolation is then invoked to create a scalar-valued function over  $\mathbf{R}^3$ . The desired surface is simply the set of all points at which this scalar function takes on the value zero. Figure 2 (left) shows a surface that was created in this fashion by placing four zero-valued constraints at the vertices of a regular tetrahedron and placing a single interior constraint in the center of the tetrahedron. The resulting implicit surface is nearly spherical.

Figure 2 (right) shows a recursive branching object that is an interpolating implicit surface. The basic building block of this object is a triangular prism. Each of the six vertices of a large prism specified the location of a zero-valued constraint, and a single interior constraint was placed in the center of this prism. Next, three smaller and slightly tilted prisms were placed atop the first large prism. Each of these smaller prisms, like the large one, contributes boundary constraints at its vertices and has a

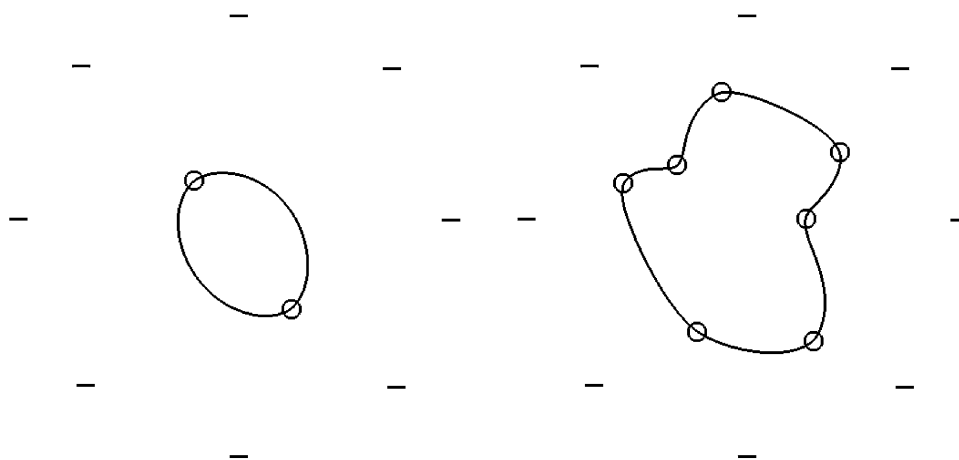


Fig. 4. Curves defined using surrounding exterior constraints. Just two zero-valued constraints yield an ellipse-like curve (on the left). More constraints create a more complex curve (at right).

single interior constraint placed at its center. Each of the three smaller prisms had even smaller prisms placed on top of them, and so on.

Why does this method of creating an implicit function create a smooth surface? We are creating the scalar-valued function in 3D that matches our constraints, and that minimizes a 3D energy functional similar to Equation 3. This energy functional selects a smoothly changing implicit function that matches the constraints. The iso-surface that we extract from such a smoothly changing function will almost always be smooth as well. It is *not* the case in general, however, that this iso-surface is also the minimum of a curvature-based functional over surfaces. Satisfying the 3D energy functional does not give any guarantee about the smoothness of the resulting 2D surface.

Placing one or more positive-valued constraints on the interior of a shape is an effective method of defining interpolating implicit surfaces when the shape one wishes to create is well-defined. We have found, however, that there is another approach that is even more flexible for interactive free-form surface sculpting.

## 4.2 Exterior Constraints

Figure 4 illustrates a second approach to creating interpolating implicit functions. Instead of placing positive-valued constraints inside a shape, negative-valued constraints can be placed on the exterior of the shape that is being created. We call each such negative-valued constraint an *exterior constraint*. As before, zero-valued constraints specify locations through which the implicit curve will pass. In Figure 4 (left), eight exterior constraints surround the region at which a curve is being created. As with positive-valued constraints, the magnitude of the values is unimportant, and we use the value-negative one. These exterior constraints, coupled with the curvature-minimizing nature of variational method, induce the interpolation function to take on positive values interior to the shape outlined by the zero-valued constraints. Even specifying just two boundary constraints defines a reasonable closed curve, as shown by the ellipse-like curve at the left in Figure 4. More boundary constraints result in a more complex curve, as shown on the right in Figure 4.

We have found that creating a circle or sphere of negative-valued constraints is the approach that is best suited to interactive free-form design of curves and surfaces. Once these exterior constraints are defined, the user is free to place boundary constraints in any location interior to this cage of exterior constraints. Section 5 describes the use of exterior constraints for interactive sculpting.

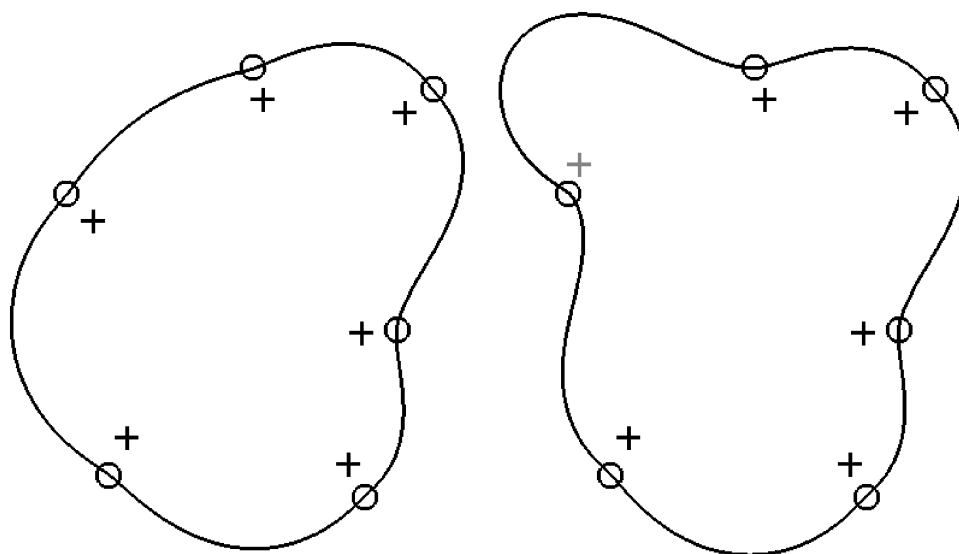


Fig. 5. Two curves defined using nearly identical boundary and normal constraints. By moving just a single normal constraint (the north-west one, shown in red), the curve on the left is changed to that shown on the right.

### 4.3 Normal Constraints

For some applications we may have detailed knowledge about the shape that is to be modeled. In particular, we may know approximate surface normals at many locations on the surface to be created. In this case there is a third method of defining an interpolating implicit function that may be preferred over the two methods described above, and this method was originally described in Turk and O'Brien [1999]. Rather than placing positive or negative values far from the boundary constraints, we can create constraints very close to the boundary constraints. Figure 5 shows this method in the plane. In the left portion of this figure, there are six boundary constraints and in addition there are six *normal constraints*. These normal constraints are positive-valued constraints that are placed very near the boundary constraints, and they are positioned towards the center of the shape that is being created. A normal constraint is created by placing a positive constraint a small distance in the direction  $-\mathbf{n}$ , where  $\mathbf{n}$  is an approximate normal to the shape that we are creating. (Alternatively, we could choose to place negative-valued constraints in the outward-pointing direction.) A normal constraint is always paired with a boundary constraint, although not every boundary constraint requires a normal constraint. The right part of Figure 5 shows that a normal constraint can be used to bend a curve at a given point.

There are at least two ways in which a normal constraint might be defined. One is to allow a user to hand-specify the surface normals of a shape that is being created. A second allows us to create smooth surfaces based on polyhedral models. If we wish to create an interpolating implicit surface from a polyhedral model, we simply need to create one boundary constraint and one normal constraint for each vertex in the polyhedron. The location of a boundary constraint is given by the position of the vertex, and the location of a normal constraint is given by moving a short distance in a direction opposite to the surface normal at the vertex. We place normal constraints 0.01 units from the corresponding boundary constraints for objects that fit within a unit cube. Figure 6 (right) shows an interpolating implicit surface created in the manner just described from the polyhedral model in Figure 6 (left). This is a simple yet effective way to create an everywhere smooth analytically defined surface. This stands in contrast to

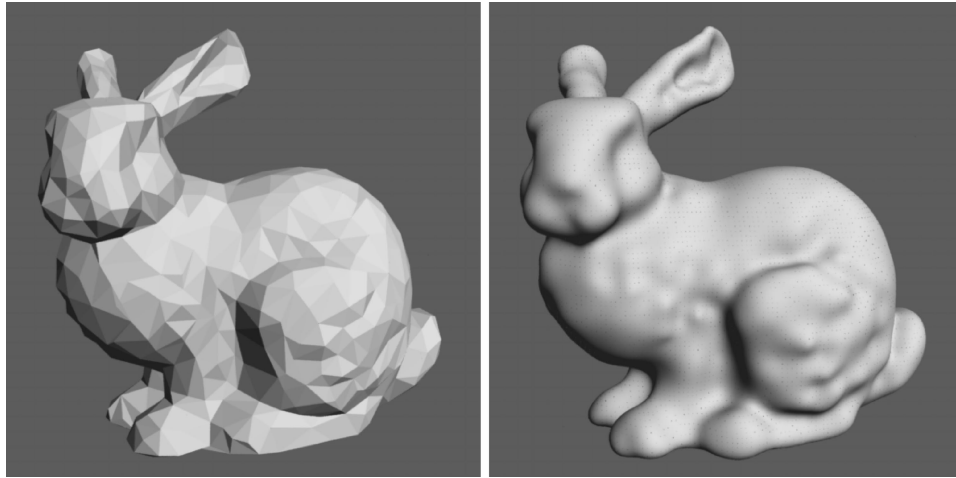


Fig. 6. A polygonal surface (left) and the interpolating implicit surface defined by the 800 vertices and their normals (right).

Table I. Constraint Types

Constraint Types	When to Use	2D Figure	3D Figure
Interior constraints	Planned model construction	Figure 1	Figure 2
Exterior constraints	Interactive modelling	Figure 4	Figures 7, 8, 10
Normal constraints	Conversion from polygons	Figure 5	Figures 3, 6, 9

the complications of patch stitching inherent in most parametric surface modeling approaches. Figure 3 is another example of converting polygons (a fist) to an implicit surface.

#### 4.4 Review of Constraint Types

In this section we have seen three methods of creating interpolating implicit functions. These methods are in no way mutually exclusive, and a user of an interactive sculpting program could well use a mixture of these three techniques to define a single surface. Table I lists each of the three kinds of constraints, when we believe each is appropriate to use, and which figures in this paper were created using each of the methods.

### 5. INTERACTIVE MODEL BUILDING

Interpolating implicit surfaces seem ready-made for interactive 3D sculpting. In this section we will describe how they can be gracefully incorporated into an interactive modeling program.

In 1994, Andrew Witkin and Paul Heckbert presented an elegant method for interactive manipulation of implicit surfaces [Witkin and Heckbert 1994]. Their method uses two types of oriented particles that lie on the surface of an implicitly defined object. One class of particles, the floaters, are passive elements that are attracted to the surface of the shape that is being sculpted. Floaters repel one another in order to evenly cover the surface. Even during large changes to the surface, a nearly constant density of floaters is maintained by particle fissioning and particle death. A second type of particle, the control point, is the method by which a user interactively shapes an implicit surface. Control points provide the user with direct control of the surface that is being created. A control point tracks a 3D cursor position that is moved by the user, and the free parameters of the implicit function are adjusted so that the surface always passes exactly through the control point. The mathematical machinery needed

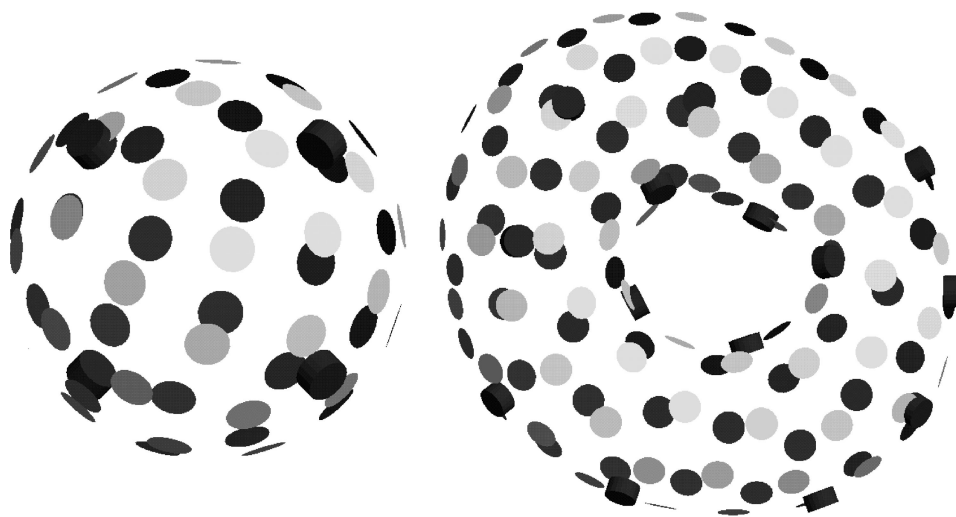


Fig. 7. Interactive sculpting of interpolating implicit surfaces. The left image shows an initial configuration with four boundary constraints (the red markers). The right surface is a sculpted torus.

to implement floaters and control points is presented clearly in Witkin and Heckbert's paper, and the interested reader should consult it for details.

The implicit surfaces used in Witkin and Heckbert's modeling program are blobby spheres and blobby cylinders. We have created an interactive sculpting program based on their particle sampling techniques, but we use interpolating implicit surfaces instead of blobbies as the underlying shape description. Our implementation of floaters is an almost verbatim transcription of their equations into code. The only change needed was to represent the implicit function as a sum of  $\phi(\mathbf{x}) = |\mathbf{x}|^3$  radial basis functions and to provide an evaluation routine for this function and its gradient. Floater repulsion, fissioning, and death, work for interpolating implicits just as well as when using blobby implicit functions. As in the original system, the floaters provide a means of interactively viewing an object during editing that may even change the topology of the surface.

The main difference between our sculpting system and Witkin and Heckbert's is that we use an entirely different mechanism for direct interaction with a surface. Witkin/Heckbert control points provide an *indirect* link between a 3D cursor and the free parameters of a blobby implicit function. We do *not* make use of Witkin and Heckbert's control particles in our interactive modelling program. Instead, we simply allow users to create and move the boundary constraints of an interpolating implicit surface. This provides a direct way to manipulate the surface.

We initialize a sculpting session with a simple interpolating implicit surface that is nearly spherical; this is shown at the left in Figure 7. It is described by four boundary constraints at the vertices of a unit tetrahedron (the thick red disks) and with eight exterior (negative) constraints surrounding these at the corners of a cube with a side width of six. (The exterior constraints are not drawn.) A user is free to drag any of the boundary constraint locations using a 3D cursor, and the surface follows. The user may also create any number of new boundary constraints on the surface. The location of a new boundary constraint is found by intersecting the surface with a ray that passes through the camera position and the cursor. After a user creates or moves a boundary constraint, the matrix equation from Section 3 is solved anew. The floaters are then moved and displayed. The right portion of Figure 7 shows a toroidal surface that was created using this interactive sculpting paradigm. The interactive program repeatedly

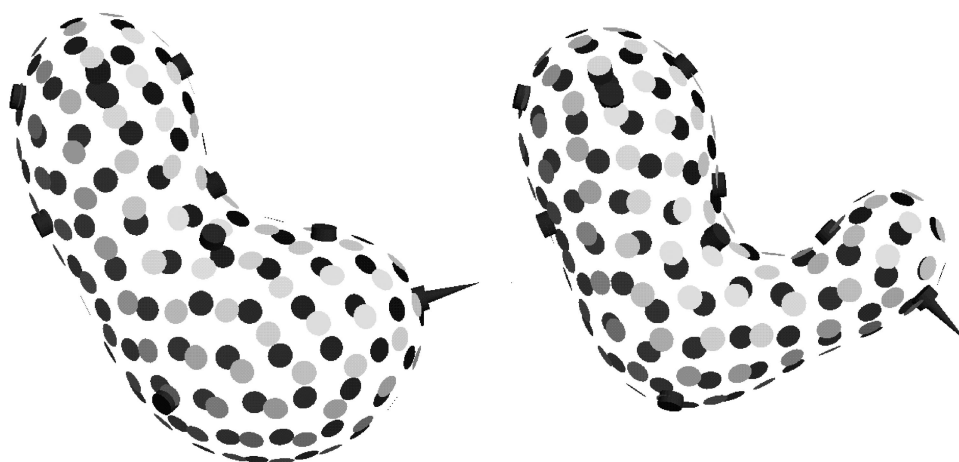


Fig. 8. Changing a normal constraint. Left image shows the original surface, and right image shows the same surface after changing a normal constraint (shown as a red spike).

executes the following steps:

1. Create or move constraints based on user interaction.
2. Solve new variational matrix equation.
3. Adjust floater positions (with floater birth and death).
4. Render floaters.

An important consequence of the matrix formulation given by Equation 8 is that adding a new boundary constraint on the existing surface does not affect the surface shape at all. This is because the implicit function already takes on the value of zero at the surface, so adding a new zero-valued constraint on the surface will not alter the surface. Only when such a new boundary constraint is moved, does it begin to affect the shape of the surface. This ability to retain the exact shape of a surface while adding new boundary constraints is similar in spirit to knot insertion for polynomial spline curves and surfaces. We do not know of any similar capability for blobby implicit surfaces.

In addition to control of boundary constraints, we also allow a user to create and move normal constraints. By default, no normal constraint is provided for a newly created boundary constraint. At the user's request, a normal constraint can be created at any specified boundary constraint. The initial direction of the normal constraint is given by the gradient of the current implicit function. The value for such a constraint is given by the implicit function's value at the constraint location. A normal constraint is drawn as a spike that is fixed at one end to the disk of its corresponding boundary point. The user may drag the free end of this spike to adjust the normal to the surface, and the surface follows this new constraint. Figure 8 shows an example of changing a normal constraint during an interactive modelling session.

What has been gained by using interpolating implicit functions instead of blobby spheres and cylinders? First, the interpolating implicit approach is easier to implement because the optimization machinery for control points of blobby implicits is not needed. Second, the user has control over the surface normal as well as the surface position. Finally, the user does not need to specify which implicit parameters are to be fixed and which are to be free at different times during the editing session. Using the blobby formulation, the user must choose, at any given time, which parameters such as sphere centers, radii of influence, and cylinder endpoints may be altered by moving a control point. With the variational

formulation, the user is always changing the position of just a single boundary or normal constraint. We believe that this direct control of the parameters of the implicit function is more natural and intuitive. Witkin and Heckbert [1994] state the following:

Another result of this work is that we have discovered that implicit surfaces are slippery:  
*when you attempt to move them using control points they often slip out of your grasp.*  
 (emphasis from the original paper)

In contrast to blobby implicits, we have found that *interpolating* implicit surfaces are not at all slippery. Users easily grasp and re-shape these surfaces with no thought to the underlying parameters of the model.

## 6. OBJECT BLENDING

A *blend* is a portion of a surface that smoothly joins two sub-parts of an object. One of the more useful attributes of implicit surfaces is the ease with which they allow two objects to be blended together. Simply summing together the implicit functions for two objects often gives quite reasonable results for some applications. In some instances, however, traditional implicit surface methods have been found to be problematic when creating certain kinds of blends. For example, it is difficult to get satisfactory results when summing together the implicit functions for two branches and a trunk of a tree. The problem is that the surface will bulge at the location where the trunk and the two branches join. Bulges occur because the contribution of multiple implicit functions causes their sum to take on large values in the blend region, and this results in the new function reaching the iso-surface threshold in locations further away from the blend than is desirable. Several solutions have been proposed for this problem of bulges in blends, but these methods are either computationally expensive or are fairly limited in the geometry for which they can be used. For an excellent description of various blending methods, see Chapter 7 of Bloomenthal [1997].

Interpolating implicit surfaces provide a new way in which to create blends between objects. Objects that are blended using this new approach are free of the bulging problems found using some other methods. Our approach to blending together surfaces is to form one large collection of constraints by collecting together the constraints that define all the surfaces to be blended. The new blended surface is the surface defined by this new collection of constraints. It is important to note that simply using *all* of the constraints from the original surfaces will usually produce poor results. The key to the success of this approach is to throw out those constraints that would cause problems.

Consider the task of blending together two shapes  $A$  and  $B$ . If we used all of the constraints from both shapes, the resulting surface is not likely to be what we wish. The task of selecting which constraints to keep is simple. Let  $f_A(\mathbf{x})$  and  $f_B(\mathbf{x})$  be the implicit functions for shapes  $A$  and  $B$  respectively. We will retain those constraints from object  $A$  that are outside of  $B$ . That is, a constraint from  $A$  with position  $\mathbf{c}_i$  will be kept if  $f_B(\mathbf{c}_i) < 0$ . All other constraints from  $A$  will be discarded. Likewise, we will keep only those constraints from object  $B$  that are outside of object  $A$ . To create a blended shape, we collect together all of the constraints that pass these two tests and form a new surface based on these constraints.

This approach can be used to blend together any number of objects. Figure 9 (left) shows three polygonal tori that overlap one another in 3D. To blend these objects together, we first create a set of boundary and normal constraints for each object, using the approach described in Section 4.3. We then keep only those constraints from each object that are outside of each of the other two objects, as determined by their implicit functions. Finally, we create a single implicit function using all of the constraints from the three objects that were retained. Figure 9 (right) shows the result of this procedure. Notice that there are no bulges in the locations where the tori meet.



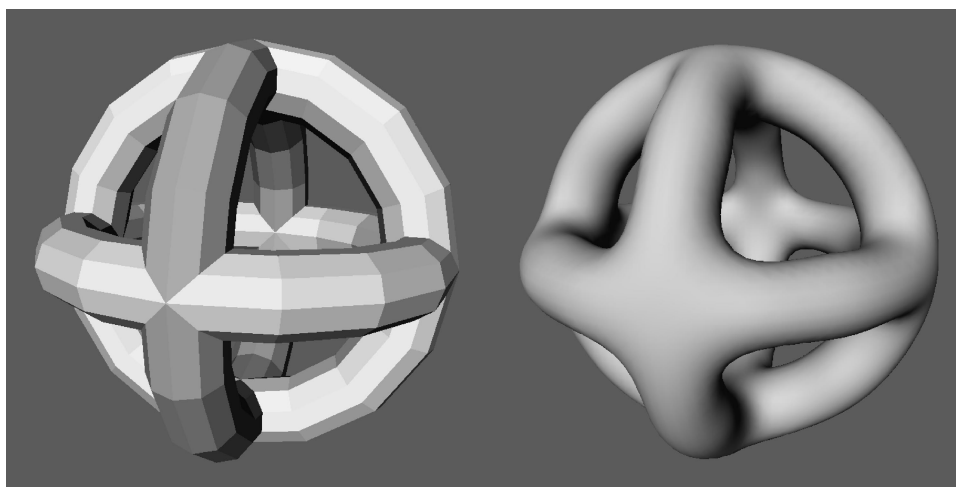


Fig. 9. Three polygonal tori (left), and the soft union created with interpolating implicits (right).

## 7. RENDERING

In this section we examine two traditional approaches for rendering implicit surfaces that both perform well for interpolating implicits.

### 7.1 Conversion to Polygons

One way to render an implicit surface is to create a set of polygons that approximate the surface and then render these polygons. The topic of iso-surface extraction is well-studied, especially for regularly sampled volumetric data. Perhaps the best known approach of this type is the Marching Cubes algorithm [Lorensen and Cline 1987], but a number of variants of this method have been described since the time of its publication.

We use a method of iso-surface extraction known as a *continuation* approach [Bloomenthal 1988] for many of the figures in this paper. The models in Figure 2 and in the right images of Figures 6 and 9 are collections of polygons that were created using the continuation method. This method first locates any position that is on the surface to be tiled. This first point can be thought of as a single corner of a cube that is one of an infinite number of cubes in a regular lattice. The continuation method then examines the values of the implicit function at neighboring points on the cubic lattice and creates polygons within each cube that the surface must pass through. The neighboring vertices of these cubes are examined in turn, and the process eventually crawls over the entire surface defined by the implicit function. We use the implementation of this method from Bloomenthal [1994] that is described in detail in Bloomenthal [1988].

### 7.2 Ray Tracing

There are a number of techniques that may be used to ray trace implicit surfaces, and a review of these techniques can be found in Hart [1993]. We have produced ray traced images of interpolating implicit surfaces using a particular technique introduced by Hart [1997] that is known as *sphere tracing*. Sphere tracing is based on the idea that we can find the intersection of a ray with a surface by traveling along the ray in steps that are small enough to avoid passing through the surface. At each step along the ray the method conservatively estimates the radius of a sphere that will not intersect the surface. We declare that we are near enough to the surface when the value of  $f(\mathbf{x})$  falls below some tolerance  $\epsilon$ . We

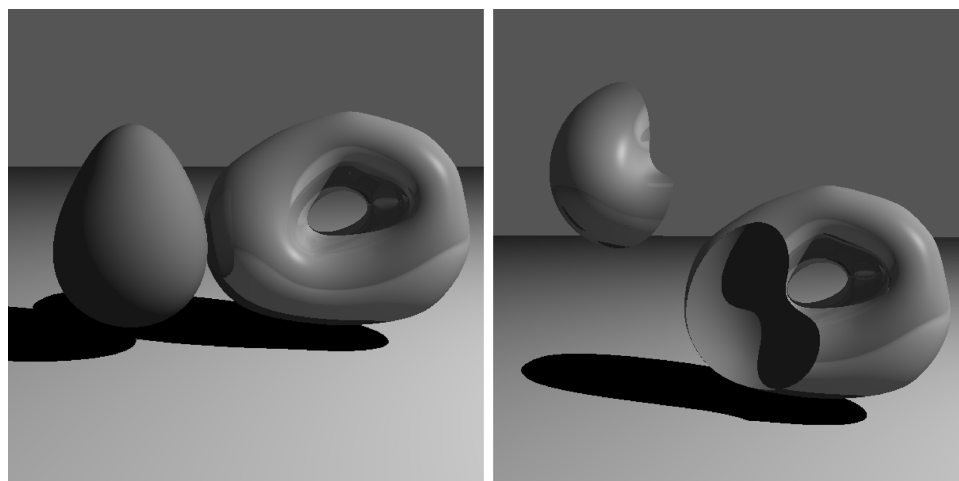


Fig. 10. Ray tracing of interpolating implicit surfaces. The left image shows reflection and shadows of two implicit surfaces, and the right image illustrates constructive solid geometry.

currently use a heuristic to determine the radius of the spheres during ray tracing. We sample the space in and around our implicit surface at 2000 positions, and we use the maximum gradient magnitude over all of these locations as the Lipschitz constant for sphere tracing. For extremely pathological surfaces, this heuristic may fail, although it has worked well for all of our images. Coming up with a sphere radius that is guaranteed not to intersect the surface is a good area for future research. We think it is likely that other ray tracing techniques can also be successfully applied to ray tracing of interpolating implicits, such as the LG-surfaces approach of Kalra and Barr [1989].

Figure 10 (left) is an image of two interpolating implicit surfaces that were ray traced using sphere tracing. Note that this figure includes shadows and reflections. Figure 10 (right) illustrates constructive solid geometry with interpolating implicit surfaces. The figure shows (from left to right) intersection and subtraction of two implicit surfaces. This figure was created using standard ray tracing CSG techniques as described in Roth [1982].

The rendering techniques of this section highlight a key point—interpolating implicit surfaces may be used in almost all of the contexts in which other implicit formulations have been used. This new representation may provide fruitful alternatives for a number of problems that use implicit surfaces.

## 8. COMPARISON TO RELATED METHODS

At this point it is useful to compare interpolating implicit surfaces to other representations of surface geometry. Although they share similarities with existing techniques, interpolating implicits are distinct from other forms of surface modeling. Because interpolating implicits are not yet well known, we provide a comparison of them to two more well-known modelling techniques.

### 8.1 Thin-Plate Surface Reconstruction

The scientific and engineering literature abound with surface reconstruction based on thin-plate interpolation. Aren't interpolating implicits just a slight variant on thin-plate techniques? The most important difference is that traditional thin-plate reconstruction creates a *height field* in order to fit a given set of data points. The use of a height field is a barrier towards creating closed surfaces and surfaces of arbitrary topology. For example, a height field cannot even represent a simple sphere-like

object such as the surface shown in Figure 2 (left). Complex surfaces can be constructed using thin-plate techniques only if a number of height fields are stitched together to form a parametric quilt over the surface. This also pre-supposes that the topology of the shape to be modelled is already known. Interpolating implicit surfaces, on the other hand, do not require multiple patches in order to represent a complex model. Both methods create a function based on variational methods, but they differ in the dimension of the scalar function that they create. Traditional thin-plate surfaces use a function with a 2D domain to create a *parametric* surface, whereas the interpolating implicit method uses a function with a 3D domain to specify the location of an *implicit* surface.

## 8.2 Sums of Implicit Primitives

Section 3 shows that an interpolating implicit function is in fact a sum of a number of functions that have radial symmetry (based on the  $|\mathbf{x}|^3$  function). Isn't this similar to constructing an implicit function by summing a number of spherical Gaussian functions (blobby spheres or meta-balls)? Let us consider the process of modeling a particular shape using blobby spheres. The unit of construction is the single sphere, and two decisions must be made when we add new sphere to a model: the sphere's center and its radius. We cannot place the center of the sphere where we want the surface to be—we must displace it towards the object's center and adjust its radius to compensate for this displacement. What we are doing is much like guessing the location of the medial axis of the object that we are modeling. (The medial axis is the locus of points that are equally distant from two or more places on an object's boundary.) In fact, the task is more difficult than this because summing multiple blobby spheres is not the same as calculating the union of the spheres. The interactive method of Witkin and Heckbert [1994] relieves the user from some of this complexity, but still requires the user to select which blobby primitives are being moved and which are fixed. These issues never come up when modeling using interpolating implicit surfaces because we can directly specify locations that the surface must pass through.

Fitting blobby spheres to a surface is an art, and indeed many beautiful objects have been sculpted in this manner. Can this process be entirely automated? Muraki [1991] demonstrated a way in which a given range image may be approximated by blobby spheres. The method begins with a single blobby sphere that is positioned to match the data. Then the method repeatedly selects one blobby sphere and splits it into two new spheres, invoking an optimization procedure to determine the position and radii of the two spheres that best approximates the given surface. Calculating a model composed of 243 blobby spheres "took a few days on a UNIX workstation (Stardent TITAN3000 2 CPU)." Similar blobby sphere data approximation by Bittar et al. [1999] was limited to roughly 50 blobby spheres. In contrast to these methods, the bunny in Figure 6 (right) is an interpolating implicit surface with 800 boundary and 800 normal constraints. It required 1 minute 43 seconds to solve the matrix equation for this surface, and the iso-surface extraction required 7 minutes 43 seconds. Calculations were performed on an SGI O2 with a 195 MHz R10000 processor.

## 9. CONCLUSION AND FUTURE WORK

In this paper we have introduced new approaches for model creation using interpolating implicit surfaces. Specific advantages of this method include:

- Direct specification of points on the implicit surface
- Specification of surface normals
- Conversion of polygon models to smooth implicit forms
- Intuitive controls for interactive sculpting
- Addition of new control points that leave the surface unchanged (like knot insertion)
- A new approach to blending objects

A number of techniques have been developed for working with implicit surfaces. Many of these techniques could be directly applied to interpolating implicits, indicating several directions for future work. The critical point analysis of Stander and Hart [1997] could be used to guarantee topologically correct tessellation of such surfaces. Interval techniques, explored by Duff, Snyder and others, might be applied to tiling and ray tracing of interpolating implicits [Duff 1992; Snyder 1992]. The interactive texture placement methods of Pedersen [1995; 1996] should be directly applicable to interpolating implicit surfaces. Finally, many marvelous animations have been produced using blobby implicit surfaces [Blinn 1982; Wyvill et al. 1986]. We anticipate that the interpolating properties of these implicit surfaces may provide animators with an even greater degree of control over implicit surfaces.

Beyond extending existing techniques for this new form of implicit surface, there are also research directions that are suggested by issues that are specific to our technique. Like blobby sphere implicits, interpolating implicit surfaces are everywhere smooth. Perhaps there are ways in which sharp features such as edges and corners can be incorporated into an interpolating implicit model. We have shown how gradients of the implicit function may be specified indirectly, using positive constraints that are near zero constraints, but it may be possible to modify the approach to allow the exact specification of the gradient.

Another direction for future research is to find higher-level interactive modelling techniques for creating these implicit surfaces. Perhaps several new constraints could be created simultaneously, maybe arranged in a line or in a circle for greater surface control. It might also make sense to be able to move the positions of more than one constraint at a time. Another modelling issue is the creation of surfaces with boundaries. Perhaps a second implicit function could specify the presence or absence of a surface. Another issue related to interactivity is the possibility of displaying the surface with polygons rather than with floaters. With sufficient processor power, creating and displaying a polygonal isosurface of the implicit function could be done at interactive rates.

#### ACKNOWLEDGMENTS

We thank the members of the Georgia Tech Geometry Group for their ideas and enthusiasm. Also thanks to Victor Zordan for help with video. Finally, we are grateful to the reviewers for their suggestions to improve this paper.

#### REFERENCES

- BITTAR, E., TSINGOS, N., AND GASCUEL, M.-P. 1995. Automatic reconstruction of unstructured 3D data: Combining a medial axis and implicit surfaces. *Computer Graphics Forum (Proceedings of Eurographics '95)* 14, 3, 457–468.
- BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Trans. Graph.* 1, 3, 235–256.
- BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Computer-Aided Geometric Design* 5, 4, 341–355.
- BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press, Cambridge, 324–349.
- BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.
- CARR, J. C., MITCHELL, T. J., BEATSON, R. K., CHERRIE, J. B., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 67–76.
- CELNIKER, G. AND GOSSARD, D. 1991. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics (SIGGRAPH 91)* 25, 4 (July), 257–266.
- DUCHON, J. 1977. Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions on Several Variables, Lecture Notes in Mathematics* 571, W. Schempp and K. Zeller, Eds. Springer-Verlag, Berlin.
- DUFF, T. 1992. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (SIGGRAPH 92)* 26, 2 (July), 154–168.
- DYN, N. 1987. Interpolation of scattered data by radial basis functions. In *Topics in Multivariate Approximation*, L. L. S. C. K. Chui and F. I. Utreras, Eds. Academic Press, Cambridge, 47–61.

- GIROSI, F., JONES, M., AND POGGIO, T. 1993. Priors, stabilizers and basis functions: from regularization to radial, tensor and additive splines. Tech. rep., MIT Artificial Intelligence Laboratory. June. A.I. Memo No. 1430.
- GRIMSON, W. E. L. 1983. Surface consistency constraints in vision. *Computer Vision, Graphics, and Image Processing* 24, 1 (Oct.), 28–51.
- HART, J. 1993. Ray tracing implicit surfaces. *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, 1–16.
- HART, J. 1997. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10, 527–545.
- KALRA, D. AND BARR, A. 1989. Guaranteed ray intersection with implicit surfaces. *Computer Graphics (SIGGRAPH 89)* 23, 4, 297–306.
- KEREN, D. AND GOTSMAN, C. 1998. Tight fitting of convex polyhedral shapes. *Int. J. Shape Modeling*, 111–126.
- LORENSEN, W. AND CLINE, H. E. 1987. Marching cubes: A high resolution 3-D surface construction algorithm. *Computer Graphics (SIGGRAPH 87)* 21, 4 (July), 163–169.
- MIRAKI, S. 1991. Volumetric shape description of range data using ‘blobby model’. *Computer Graphics (SIGGRAPH 91)* 25, 4 (July), 227–235.
- MORSE, B., YOO, T. S., RHEINGANS, P., CHEN, D. T., AND SUBRAMANIAN, K. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modelling International*, 89–98.
- NISHIMURA, H., HIRAI, M., KAWAI, T., KAWATA, T., SHIRKAWA, I., AND OMURA, K. 1985. Object modeling by distribution function and a method of image generation. *Trans. Inst. Elect. Commun. Eng. Japan J68-D*, 4, 718–725.
- PEDERSEN, H. 1995. Decorating implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 95)*, 291–300.
- PEDERSEN, H. 1996. A framework for interactive texturing on curved surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 96)*, 295–302.
- ROTH, S. 1982. Ray casting as a method for solid modeling. *Computer Graphics and Image Processing* 18, 2, 109–144.
- SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNNI, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum* 14, 4 (Oct.), 181–188.
- SNYDER, J. 1992. Interval analysis for computer graphics. *Computer Graphics (SIGGRAPH 92)* 26, 2 (July), 121–130.
- STANDER, B. T. AND HART, J. C. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 97)*, 279–286.
- SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *IEEE Trans. Pattern Anal. Mach. Intell.* 12, 6 (June), 513–528.
- TAUBIN, G. 1993. An improved algorithm for algebraic curve and surface fitting. In *Fourth International Conference on Computer Vision (ICCV ’93)*. IEEE, Berlin, Germany, 658–665.
- TERZOPOULOS, D. 1988. The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell.* 10, 4 (July), 417–438.
- TURK, G. AND O’BRIEN, J. 1999. Shape transformation using variational implicit functions. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1999)*, 335–342.
- WELCH, W. AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, 247–256.
- WITKIN, A. P. AND HECKBERT, P. S. 1994. Using particles to sample and control implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, 269–278.
- WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structures for soft objects. *The Visual Computer* 2, 4, 227–234.

Received July 2001; revised April 2002; accepted May 2002

# Some Notes on Radial Basis Functions and Thin Plate Splines

John C. Hart  
Dept. of Computer Science  
University of Illinois Urbana-Champaign

April 24, 2005

Many methods exist for defining a scalar field  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  given  $N$  sample values  $f_i \in \mathbb{R}$  at  $N$  scattered data points  $\mathbf{x}_i \in \mathbb{R}^n$ . These methods largely construct the scalar field as the linear combination of  $N$  basis functions  $f(\mathbf{x}) = \sum_{i=1}^N a_i \phi_i(\mathbf{x})$ . For example, Fourier and wavelet methods transform these data points into a corresponding number of frequencies and amplitudes (and phases). But a theorem of Haar shows that some configurations of data points can lead to ill conditioned and even degenerate linear combinations.

A more robust approach arises from so-called data dependent methods whose basis functions depend on the locations of the data points. These basis functions are often radially symmetric and centered at each of the data points, taking the form

$$f(\mathbf{x}) = \sum_{i=1}^N a_i \phi(\|\mathbf{x} - \mathbf{x}_i\|). \quad (1)$$

## 1 Shepard's Method

An early method in this form is Shepard's method, which is based on the radially-symmetric basis functions of invese power distance  $1/r^\mu$  where typically  $\mu = 1, 2$ . Normalizing these basis functions to create a partition of unity and then weighting each basis function by the desired value  $f_i$  of its corresponding point  $\mathbf{x}_i$  yields

$$f(\mathbf{x}) = \frac{\sum_{i=1}^N f_i \|\mathbf{x} - \mathbf{x}_i\|^{-\mu}}{\sum_{i=1}^N \|\mathbf{x} - \mathbf{x}_i\|^{-\mu}}. \quad (2)$$

which when divided out results in the form (1) with the radially-asymmetric basis functions

$$\phi_i(\mathbf{x}) = \frac{\prod_{j \neq i} \|\mathbf{x} - \mathbf{x}_j\|^\mu}{\sum_{i=1}^N \prod_{j \neq i} \|\mathbf{x} - \mathbf{x}_j\|^\mu}. \quad (3)$$

That  $f(\mathbf{x}_i) = f_i$  can be verified by showing  $\phi_i(\mathbf{x}_j) = \delta_{ij}$ .

Shepard's method is fast since it does not require solving a linear system to find the basis weights, instead using the data point's desired valued  $f_i$  directly. However, the quality of the surface reconstructed by Shepard's method suffers near the data points, with corners when  $\mu = 1$  or flat regions when  $\mu \geq 2$ . These artifacts can be reduced by interpolating  $f_i + \nabla f \cdot (\mathbf{x} - \mathbf{x}_i)$  instead of  $f_i$  but this sacrifices the accuracy of the surface reconstruction to overcome a problem in the formulation. A version of Shepard's method derived from quadratic B-spline kernels organized in an octree hierarchy was used recently to create the multi-level partition of unity (MPU) method [3].

## 2 Radial Basis Functions

Shepard's method avoids the careful choice in (1) of weights  $a_i$  to ensure  $f(\mathbf{x}_i) = \sum a_i \phi_i(\mathbf{x}_i) = f_i$  by using basis functions that are not radially symmetric about the  $\mathbf{x}_i$ . If the basis functions are radially symmetric then we need to solve a linear system

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N1} & \phi_{N2} & \dots & \phi_{NN} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \quad (4)$$

where  $\phi_{ij} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|)$  to ensure the proper weights are chosen.

The complete radial basis function (RBF) method augments (1) with the addition of a polynomial  $P(\mathbf{x})$  in the components of  $\mathbf{x}$ . This polynomial term is usually linear (adding linear precision to the system) and its four coefficients are added to the matrix system and solved with the rest. This matrix can be solved in linear time using a multipole expansion [1]. The matrix is dense but compact-support radial basis functions exist (at the expense of smoothness) that yield a more efficient sparse matrix [2, 4, 5].

Radial-basis-function representations must have at least one center off the isosurface to avoid trivial (constant) solutions to (4). One strategy is to place single RBF center set to an "inside" value in the middle of an object, but it is sometimes difficult to determine the middle of a contorted object. Another strategy is to surround the object with RBF centers set to an "outside" value, but these values can sometimes interfere with the shape of the RBF surface [6].

An alternative to these inside and outside constraints is possible through the use of *dipoles* [6]. Instead of placing 0-valued RBF centers on the surface at  $\mathbf{x}_i$ , we place a pair of centers at  $\mathbf{x}_i \pm \epsilon \mathbf{n}$  with values  $\pm \epsilon$ , where  $\mathbf{n}_i = \nabla \phi_i / |\nabla \phi_i|$  is the RBF surface normal at  $\mathbf{x}_i$ . In addition to overcoming the previous problems, RBF dipoles have the additional benefit of exerting control over the local orientation of the RBF surface.

## 2.1 Thin Plate Splines

(The following section began with the notes of USC's J.P. Lewis provided by Matthieu Desbrun, which were subsequently corrected and rederived this past summer with the help of UIUC mathematician Rob Ghrist and his grad student Jaebum Jung in preparation for a shape modeling class this past fall.)

In 2-D, this interpolation results in a height field  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  where the  $\mathbf{x}_i \in \mathbb{R}^2$  are positions in the plane and the  $f_i$  are the desired altitude of  $f$  at those positions. The bending energy of a height field is the functional  $B[f] = \iint (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2) dx dy$ . If the radial basis function  $\phi(r) = r^2 \log r$  is used, then the resulting height field is a thin plate spline of minimum bending energy, as would be a thin sheet of metal passing through the points. In odd dimension, e.g.  $\mathbf{x}_i \in \mathbb{R}^3$ , the radial basis function  $\phi(r) = r^3$  minimizes the bending energy of the field. This is easiest to show for the 1-D case, where the points  $\mathbf{x}_i$  are the single coordinates  $x_i$ .

If we let  $P$  denote a second derivative operator, then

$$B[f] = \|Pf\|^2 = \langle Pf, Pf \rangle = \langle P^\dagger Pf, f \rangle = \langle Lf, f \rangle \quad (5)$$

using the Frobenius norm, any suitable inner product over the Hilbert space of  $f$ , the definition of the adjoint as  $\langle Ax, x \rangle = \langle x, A^\dagger x \rangle$ , and the fourth derivative operator  $L$  as a shorthand for  $P^\dagger P$ .

In order to incorporate the minimization of bending energy into the least-squares interpolation of point data, we minimize an energy functional

$$E[f] = \sum_{i=1}^N (f(x_i) - f_i)^2 + \lambda B[f] \quad (6)$$

that combines the two using  $\lambda$  to control the balance. Its variational derivatives are

$$\frac{dE}{df}[f] = 2 \sum_{i=1}^N (f(x_i) - f_i) \delta(x - x_i) + \lambda \frac{dB}{df}[f] \quad (7)$$

$$\frac{dB}{df}[f] = \lim_{a \rightarrow 0} \frac{2a \langle Pf, Pg \rangle + a^2 \langle Pg, Pg \rangle}{a} = 2 \langle Lf, g \rangle \quad (8)$$

where  $g$  is a test function. Setting (7) to zero yields

$$Lf = -\frac{1}{\lambda} \sum_{i=1}^N (f(x_i) - f_i) \delta(x - x_i). \quad (9)$$

We “divide by  $L$ ” on both sides by finding the Green's function  $G$  of  $L$  such that  $LG(x, x_i) = \delta(x - x_i)$ . This function is a “convolutional inverse” of  $L$  such that

$$f = G \star \left( -\frac{1}{\lambda} \sum_{i=1}^N (f(x_i) - f_i) \delta(x - x_i) \right). \quad (10)$$



It is easier to work with convolution in the Fourier domain, where it becomes a simple product. Recall the Fourier transform of the derivative of a function

$$\mathcal{F}[f'(t)] = \int_{-\infty}^{\infty} f'(t)e^{-i\omega t} dt = f(t)e^{i\omega t} \Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} f(t)(-i)\omega e^{-i\omega t} dt = i\omega \int_{-\infty}^{\infty} f(t)e^{-i\omega t} dt \quad (11)$$

where the second step is the result of an integration by parts with  $v = e^{-i\omega t}$  and  $du = f'(t)dt$ , and the evaluation of the portion outside the integral goes to zero. Iterating we find the Fourier transform of the  $k$ th derivative is  $\mathcal{F}[f^{(k)}(t)] = |\omega|^k \mathcal{F}[f(t)]$ .

Recall  $L = P^\dagger P$  is a fourth derivative operator, so  $\mathcal{F}[LG] = \omega^4 \mathcal{F}[G]$ . Also  $\mathcal{F}[LG] = \mathcal{F}[\delta(x - x_0)]$  and the latter is a constant, specifically  $e^{-i\omega x_0}$ . Hence  $\mathcal{F}[G] \propto 1/\omega^4$  which corresponds to the function  $|x|^3$  in the spatial domain. Computing  $\mathcal{F}^{-1}(1/\omega^4)$  is difficult because the integral diverges, but can be windowed with a factor of  $e^{-a|x|}$  and taking the limit as  $a \rightarrow 0$ ,

$$\mathcal{F}[e^{-a|x|}|x|^3] = \int e^{-a|x|}|x|^3 e^{-i2\pi\omega x} dx \quad (12)$$

$$= \int_0^{\infty} e^{-a|x|} x^3 e^{-i2\pi\omega x} dx + \int_{-\infty}^0 e^{-a|x|} (-x)^3 e^{-i2\pi\omega x} dx \quad (13)$$

$$= 6(a + i2\pi\omega)^{-4} + 6(a + i2\pi\omega)^{-4}. \quad (14)$$

The limit  $\lim_{a \rightarrow 0} \mathcal{F}[e^{-a|x|}|x|^3]$  gives us  $\lim_{a \rightarrow 0} 12(a + i2\pi\omega)^{-4}$  which is proportional to  $1/\omega^4$ .

## References

- [1] CARR, J., BEATSON, R., CHERRIE, J., MITCHELL, T., FRIGHT, W., MCCALLUM, B., AND EVANS, T. Reconstruction and representation of 3d objects with radial basis functions. *Computer Graphics (Proc. SIGGRAPH '01)* 35 (2001), 67–76.
- [2] MORSE, B., YOO, T., RHEINGANS, P., CHEN, D., AND SUBRAMANIAN, K. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proc. Shape Modeling International* (2001), pp. 89–98.
- [3] OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. Multi-level partition of unity implicits. *ACM Trans. on Graphics* 22, 3 (2003), 463–470. Proc. SIGGRAPH.
- [4] OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003* (2003), IEEE Computer Society, p. 292.
- [5] OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 3d scattered data approximation with adaptive compactly supported radial basis functions. In *Proc. Shape Modeling Intl.* (2004), pp. 31–39.
- [6] TURK, G., AND O'BRIEN, J. F. Modelling with implicit surfaces that interpolate. *ACM Trans. on Graphics* 21, 4 (2002), 855–873.

# Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling

Barton T. Stander<sup>1</sup>      John C. Hart<sup>2</sup>

School of EECS  
Washington State University

## Abstract

Morse theory shows how the topology of an implicit surface is affected by its function’s critical points, whereas catastrophe theory shows how these critical points behave as the function’s parameters change. Interval analysis finds the critical points, and they can also be tracked efficiently during parameter changes. Changes in the function value at these critical points cause changes in the topology. Techniques for modifying the polygonization to accommodate such changes in topology are given. These techniques are robust enough to guarantee the topology of an implicit surface polygonization, and are efficient enough to maintain this guarantee during interactive modeling. The impact of this work is a topologically-guaranteed polygonization technique, and the ability to directly and accurately manipulate polygonized implicit surfaces in real time.

**Descriptors:** I.3.5 Computational Geometry and Object Modeling — Modeling packages. **General Terms:** Algorithms.

**Keywords:** catastrophe theory, critical points, implicit surfaces, Morse theory, polygonization, topology, interval analysis, interactive modeling, particle systems.

## 1 Introduction

Shapes are represented implicitly by a function that classifies points in space as inside the shape, outside the shape or on the shape’s surface, called the implicit surface. This representation provides computer graphics with geometric models that can be easily and smoothly joined together, but the incorporation of the implicit representation into graphics systems can be problematic. Polygonization of implicit surfaces allows graphics systems to reap the powerful modeling benefits of the implicit representation while retaining the rendering speed and flexibility of polygonal meshes.

The *topology* (specifically the *homotopy type*) of an implicit surface refers to the connectedness of the shape, including the number of disjoint components and the number of holes in each component (*genus*). This should not be confused with other instances

of topology in computer graphics, such as the connection patterns of a mesh of polygons or parametric patches. Thus, for a polygonization to accurately represent the topology of an implicit surface the number of components and the genus of each component of the polygonization need to agree with that of the implicit surface.

Implicit surfaces are commonly polygonized to a given degree of geometric accuracy. This accuracy is in some cases adaptively related to the local curvature of the implicit surface, reducing the number of polygons without affecting the appearance of the surface. This work instead focuses on a polygonization that accurately discerns the topology of an implicit surface. Topologically-accurate polygonization can reduce the number of polygons without affecting the structure of the surface.

Guaranteeing the topology of a polygonization does not necessarily yield an accurate polygonization. A coffee cup is topologically equivalent to a torus, but the torus provides a poor representation for the geometry of a coffee cup. The topological guarantee becomes useful when coupled with a geometrically accurate polygonization scheme.

Guaranteeing the topology of an implicit surface polygonization solves several open problems in computer graphics.

**Polygonization topology is coordinate dependent.** Polygonization schemes often discern topology from point samples. Different polygonizations of the same implicit surface may differ in topology, and the same polygonization algorithm may return different topological structures depending on the coordinate system of the implicit surface, as demonstrated in Figure 1. For example, the implicit surface might appear connected when polygonized in its modeling coordinate system but could then appear disconnected when polygonized in a scene’s coordinate system.

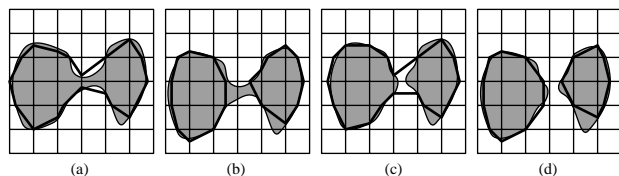


Figure 1: The topology of a connected implicit surface is correctly polygonized (a), but a translated instance is not (b). Two disjoint components are polygonized as a single component (c), but a translated instance is polygonized properly (d).

**Interloping components.** Some configurations of implicit surfaces can yield unexpected disjoint components. Such isolated components of the implicit surface occur because of an accumulation of neighboring potentials but do not themselves surround any “skele-

<sup>1</sup>Current address: Strata, 1562 El Vista Circle, Saint George, UT 84765, barts@strata3d.com

<sup>2</sup>Address: School of EECS, WSU, Pullman, WA 99164-2752, hart@eeecs.wsu.edu

tal” geometry. For example, Figure 2 shows two blobby ellipsoids that produce a third component. Such components would not appear in a continuation-based polygonization<sup>1</sup>, as might be used for modeling, but would appear as a surprise in a direct ray tracing of the implicit surface, as might be used for the final rendering.

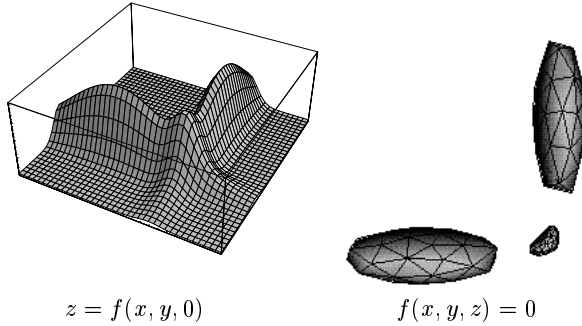


Figure 2: An interloping component found at the intersection of the potential of two blobby ellipses (left) and two polygonized blobby ellipsoids (right). The algorithms described in this paper were used to correctly polygonize the interloping configuration on the right.

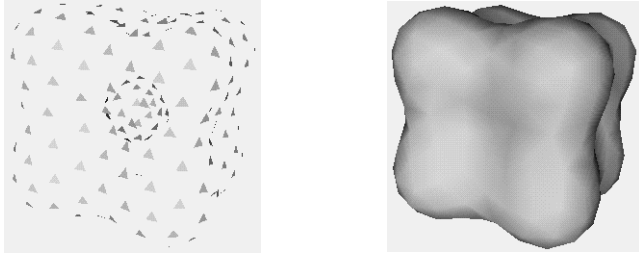


Figure 3: Implicit form displayed using a particle system (left) and polygonized (right).

**Real-time implicit surface modeling.** Implicit surfaces can be modeled and displayed in real-time using a particle system representation [35]. The viewer must then infer the shape of the implicit surface from the positions and orientations of the particles. Polygonization of the particles provides a better visual representation of the implicit surface, as demonstrated in Figure 3, but adding a polygonization step after every modeling change significantly degrades the otherwise real-time performance of the system. The ability to detect and correct topology changes allows the system to dynamically maintain the polygonization in real time by reconnecting the vertices of a polygonization only when a topology change has occurred, and only in the neighborhood of the topology change.

This work hence has two objectives. The first is to generate a polygonization of an implicit surface that is guaranteed to agree with its topology. The second objective is to maintain this topological guarantee during interactive manipulation of the surface. The topology of implicit surface polygonizations are guaranteed by tracking a few special points, called critical points, that dictate the topology of the implicit surface.

Section 2 examines previous polygonization techniques that consider topology. Section 3 describes critical points and adapts existing interval search methods and constraint techniques to the specific tasks of finding and tracking critical points. Section 4 analyzes the effect of critical points on topology and describes techniques for adjusting the topology of a polygonization to match the

<sup>1</sup>The techniques developed in this paper could be used to assist a continuation-based polygonization schemes detect such isolated components.

topology of the implicit surface. Section 5 describes a new polygonization algorithm based on Morse theory that polygonizes an implicit surface with a guarantee that the topology of the polygonization matches the topology of the implicit surface. Section 6 describes several new algorithms for maintaining this topological guarantee fast enough to support direct manipulation at interactive speeds. Section 7 concludes with a summary, implementation details and directions for further investigation.

## 2 Previous Work

One method for interrogating an implicit surface subdivides space into cells and samples the implicit surface at the corners of these cells [22, 36, 16, 2, 20].

Some cellular polygonizations can guarantee that the implicit surface is contained in the union of a set of arbitrarily small cells, hence yielding a guarantee on surface topology accurate to a given geometric precision (e.g. the diameter of the smallest cells). Both the Lipschitz condition [14] and interval analysis [30, 17] can guarantee that an implicit surface does not pass through some cells. Cells for which this guarantee fails are “ambiguous” and can be subdivided until a given level of precision is reached and the implicit surface is assumed to lie within the union of the ambiguous cells. These ambiguous cells can then be further subdivided into “globally parameterizable” components [28]. The topology of the surface passing through such a component can be determined with a few point samples. Such cellular subdivision schemes yield a geometrically precise guaranteed representation of the implicit surface topology, though at the expense of a polygonization composed of an unnecessarily large number of small polygons.

An alternative method for interrogation constrains a particle system to the implicit surface [3, 33, 31, 9, 8]. When the implicit surface remains static, such as during a pause in user manipulation, it’s particle system can be polygonized [6, 35].

A polygonization of the particle system can be maintained during user manipulation in a previous work [24]. Changes in topology were detected by comparing the interpolated polygonization normals to the implicit surface gradients at the vertices of the polygonization. Significant differences in these two vectors implied that the topology of the polygonization might not match the topology of the underlying implicit surface.

Critical points have also been used to determine implicit surface topology during a “shrinkwrap” polygonization [4]. A Lipschitz condition on the derivative of the function was used to guarantee the absence of critical points in a neighborhood, and upon failure, Newton’s method was used to find the possible critical point. If Newton’s method failed to converge, then the neighborhood was assumed to contain no critical points. The shrinkwrapping work also described a technique for repolygonizing the vertices surrounding a critical point based on the positions and orientations of the nearby polygons. Section 5 further explains and analyzes the shrinkwrap algorithm.

The analysis of topology using critical points is not entirely new to computer graphics. Critical points of vector and tensor fields are used to delineate topologically-distinct regions in the visualization of flow [13, 7]. Catastrophes have been used to understand caustics [18] and to interpret projections [15]. Morse theory has been used to reconstruct surfaces from cross sections [27] and to find surface-surface intersection curves [5].

## 3 Critical Points

The *implicit surface* defined by a function  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  is the set of points  $\mathbf{x} \in \mathbb{R}^3$  that satisfy  $f(\mathbf{x}) = 0$ . We assume the function

returns positive values inside the object, so the *solid* modeled by the implicit surface is the set of points  $\{\mathbf{x} | f(\mathbf{x}) \geq 0\}$ .

This work requires the function  $f$  to be  $C^2$  continuous, with continuous first and second derivatives, and its implicit surface must be a manifold with a well defined, continuously varying surface normal. These restrictions include exponential-based “blobby” models [1], but exclude some of the more efficient  $C^1$  piecewise polynomial approximations [21, 36].

The implicit surface is extended into a family of surfaces defined by  $f(\mathbf{x}; \mathbf{q})$  continuously parameterized by the vector  $\mathbf{q}$  consisting of various model parameters (e.g. the locations of blobby elements). For some values of  $\mathbf{q}$ , the implicit surface defined by  $f(\mathbf{x}; \mathbf{q}) = 0$  may contain a cusp, kink or crease, specifically when the implicit surface changes topology. We consider the implicit surfaces in this family before and after but not during such topology changes. An alternative technique exists for interactively manipulating implicit surfaces with cusps, kinks and creases [25].

The *critical points* of a function  $f$  occur where its gradient

$$\nabla f(\mathbf{x}) = (f_x(\mathbf{x}), f_y(\mathbf{x}), f_z(\mathbf{x})) \quad (1)$$

vanishes. (The notation  $f_x = \partial f / \partial x$ .) A *critical value* is the value of the function  $f$  at a critical point.

The *Hessian*  $V$  (called the *stability matrix* in catastrophe theory) is defined as the Jacobian of the gradient

$$V(\mathbf{x}) = J(\nabla f(\mathbf{x})) = \begin{bmatrix} f_{xx}(\mathbf{x}) & f_{xy}(\mathbf{x}) & f_{xz}(\mathbf{x}) \\ f_{yx}(\mathbf{x}) & f_{yy}(\mathbf{x}) & f_{yz}(\mathbf{x}) \\ f_{zx}(\mathbf{x}) & f_{zy}(\mathbf{x}) & f_{zz}(\mathbf{x}) \end{bmatrix}. \quad (2)$$

Since  $f_{xy} = f_{yx}$ , etc., the stability matrix is symmetric.

A critical point  $\mathbf{x}$  is classified based on the signs of the three eigenvalues  $l_1 \leq l_2 \leq l_3$  of  $V(\mathbf{x})$  [32]. If all three eigenvalues are non-zero, then the critical point is called *non-degenerate* and is either a maximum, minimum or some kind of saddle point. In three dimensions, saddle points come in two varieties. Table 1 indicates this classification.

$l_1$	$l_2$	$l_3$	Critical Point
-	-	-	Maximum Point
-	-	+	2-Saddle
-	+	+	1-Saddle
+	+	+	Minimum Point

Table 1: Classification of critical points based on the sign of the eigenvalues of the stability matrix.

The critical points are continuously dependent on the parameter vector  $\mathbf{q}$ . As the parameter vector  $\mathbf{q}$  changes, the critical points move in space. They can also appear spontaneously in pairs, or collide in pairs, annihilating each other. If any of the three eigenvalues of the stability matrix of a critical point equal zero then  $\mathbf{x}$  is called a *degenerate critical point*. The creation and destruction of critical points occur at degenerate critical points. Critical point creation/destruction is demonstrated in Figure 4.

Isolated degenerate critical points are unstable. In the rare event that an isolated degenerate critical point does appear, it can be removed by a small perturbation of the implicit surface parameters without affecting the implicit surface topology.

Some functions can yield non-isolated degenerate critical points (critical sets). For example, the cylinder defined by  $f(x, y, z) = x^2 + y^2 - 1$  has a critical line along the  $z$ -axis. A small perturbation of the cylinder into an ellipsoid  $f(x, y, z) = x^2 + y^2 + \epsilon z^2 - 1$  collapses the degenerate critical line into a single non-degenerate critical point at the origin. We assume the family of implicit surfaces is parameterized such that degenerate sets can be removed by such perturbation.

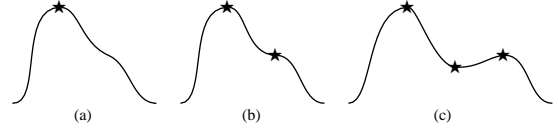


Figure 4: Creation of critical points in 1-D:  $y = f(x, 0, 0)$ . (a) Two summed Gaussian bumps, one large and one small, sufficiently close such that there is only a single maximum point in the domain shown. (b) Moving the smaller bump away from the larger creates a degenerate critical point. (c) Moving the smaller bump farther results in the creation of a pair of new critical points: a maximum point and a minimum point. Performing these steps in reverse demonstrates critical point annihilation.

### 3.1 Finding All Critical Points

Interval analysis searches can be guaranteed to find all points satisfying a given criterion in a given bounded domain to a desired degree of accuracy [19, 23]. Such a search can find all of the critical points of a given function to determine the topology of its implicit surface.

The interval search for critical points starts with an initial box bounding the space of interest. The simple interval search for critical points shown in Figure 5 eliminates large portions of space that cannot contain a critical point.

Given a box (a vector of intervals)  $\mathbf{X} = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]$  the algorithm checks whether the intervals returned by all of the partial derivatives contain zero. If not, then  $\mathbf{X}$  contains no critical points. If so, then the algorithm subdivides  $\mathbf{X}$  and tests each component individually. Note that  $F_x(\mathbf{X})$  is an interval arithmetic implementation of  $\partial f / \partial x$ , and likewise for  $F_y, F_z$ .

**Procedure SimpleSearch( $\mathbf{X}$ )**  
 If  $\text{diam}(\mathbf{X}) < \epsilon$  then indicate critical point in  $\mathbf{X}$ .  
 If  $0 \in F_x(\mathbf{X})$  and  $0 \in F_y(\mathbf{X})$  and  $0 \in F_z(\mathbf{X})$  then  
 Subdivide  $\mathbf{X}$  and continue the search recursively.

Figure 5: Simple interval divide and conquer search algorithm.

In these algorithms, subdivision means dividing into halves with respect to its widest axis, although any number of subdivision techniques could be used. The diameter of a box  $\text{diam}(\mathbf{X})$  is measured using the chessboard metric, and is simply the width of the widest interval-element in the vector  $\mathbf{X}$ .

Simple subdivision performs remarkably well, discarding large portions of space known not to contain critical points. This technique will eventually find all critical points to any degree of accuracy within a given bounding box, but with only linear convergence.

When the box diameter reaches a given size, the quadratically-convergent interval Newton’s method shown in Figure 6 refines and/or subdivides the box down to the desired numerical precision [28, 11].

Given two points  $\mathbf{x}, \mathbf{y}$  there exist points  $\mathbf{z}$  between<sup>2</sup>  $\mathbf{x}$  and  $\mathbf{y}$  such that

$$\nabla f(\mathbf{x}) + V(\mathbf{z})(\mathbf{y} - \mathbf{x}) = \nabla f(\mathbf{y}), \quad (3)$$

where the stability matrix  $V$  is the Jacobian of  $\nabla f$ . Let  $\mathbf{m}(\mathbf{X})$  return the midpoint of box  $\mathbf{X}$ . The algorithm seeks  $\mathbf{y} \in \mathbf{X}$  such that  $\nabla f(\mathbf{y}) = 0$ . Since both  $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ , the  $\mathbf{z}$  satisfying (3) must be in  $\mathbf{X}$  as well. Thus, solving

$$\nabla f(\mathbf{m}(\mathbf{X})) + V(\mathbf{X})(\mathbf{Y} - \mathbf{m}(\mathbf{X})) = 0. \quad (4)$$

yields  $\mathbf{Y}$ , a box containing all of the critical points in  $\mathbf{X}$ . Note that

```

Procedure NewtonSearch( $\mathbf{X}$ )
  Repeat.
    Solve  $V(\mathbf{X})(\mathbf{Y} - \mathbf{m}(\mathbf{X})) = -\nabla f(\mathbf{m}(\mathbf{X}))$  for  $\mathbf{Y}$ .
    If  $\mathbf{Y} \supset \mathbf{X}$  then subdivide  $\mathbf{X}$  and search recursively.
    If  $\mathbf{Y} \subset \mathbf{X}$  then there is a unique c.p. in  $\mathbf{Y}$  (and  $\mathbf{X}$ ).
    If  $\mathbf{Y} \cap \mathbf{X} = \emptyset$  then there is no c.p. in  $\mathbf{X}$ . Return.
    Otherwise let  $\mathbf{X} = \mathbf{X} \cap \mathbf{Y}$ .
  Until  $\text{diam}(\mathbf{X}) < \epsilon$ .
  Indicate critical point at  $\mathbf{m}(\mathbf{X})$ .

```

Figure 6: Interval Newton’s method search for critical points.

$V(\mathbf{X})$  returns a matrix of intervals.

An interval version of Gauss-Seidel is recommended for solving (4), but this can lead to two problems. First, the diagonal elements of  $V(\mathbf{X})$  might contain zero, requiring the interval arithmetic division operation to correctly perform a division by an interval containing zero. Division by intervals containing zero produce two intervals, leading to additional algorithm recursion [28, 10]. Solving the rows whose diagonal elements do not contain zero first reduces the occurrence of semi-infinite intervals [11].

Solving

$$V_c^{-1}V(\mathbf{X})(\mathbf{Y} - \mathbf{x}) = -V_c^{-1}\nabla f(\mathbf{m}(\mathbf{X})). \quad (5)$$

where  $V_c = m(V(\mathbf{X}))$  instead of (4) yields a much tighter bound and hastens the NewtonSearch performance [11]. Note that the expression  $m(V(\mathbf{X}))$  returns a scalar matrix consisting of the mid-points of the intervals of  $V(\mathbf{X})$ .

When a critical point lies on an edge of the box, NewtonSearch’s convergence is less quadratic. Extending  $\mathbf{X}$  outward by a small percentage each iteration avoids this problem. Time may also be incorporated into the search by crossing  $\mathbf{X}$  with the time interval  $[t_0, t_1]$ .

### 3.2 Tracking Critical Points

Altering an implicit surface’s parameters changes the positions of some or all of the critical points.

The same techniques that constrain particles to adhere to the implicit surface [35], can also cause particles to adhere to any selected critical point.

Let  $\mathbf{x} = \mathbf{x}(t)$  be a particle constrained to follow one of the critical points of the function  $f$ . Its partial derivatives  $f_x(\mathbf{x})$ ,  $f_y(\mathbf{x})$ , and  $f_z(\mathbf{x})$  are all constrained to zero. To ensure that they remain zero, their time derivatives  $\dot{f}_x(\mathbf{x}) = \frac{d^2 f}{dx dt}(\mathbf{x})$ ,  $\dot{f}_y(\mathbf{x})$  and  $\dot{f}_z(\mathbf{x})$  must also be set to zero. Given the parameter vector  $\mathbf{q}$  and its velocity  $\dot{\mathbf{q}}$ , one can solve these equations to determine the critical point velocity  $\dot{\mathbf{x}}$ . This velocity is then passed to a differential equation solver (such as fourth-order Runge-Kutta) to approximate the new location of the critical point. Newton’s method refines the approximation.

## 4 Detecting and Correcting Topology Changes

The identification of critical points simplifies topologically-guaranteed direct manipulation of implicit surfaces through a polygonal representation. The key to solving the topology problem is that a change in the topology of a surface is always accompanied by a change in the sign of a critical value [12]. Monitoring the critical points greatly simplifies the burden of detecting topological changes, and divides the problem into classifying topological changes, identifying polygons to remove and reconnecting the vertices of the removed polygons.

<sup>2</sup>The notion of “between” for points in space means that  $\mathbf{z}$  is in a box with corners at  $\mathbf{x}$  and  $\mathbf{y}$ .

### 4.1 Classifying Topological Changes

Table 2 enumerates all of the possible critical-point/sign combinations and their corresponding implications on the implicit surface topology. When an implicit surface topology change is detected, the polygonization must be altered to properly represent the new topology.

Critical Point	Sign Changes To	Action
Maximum	-	Destroy
Maximum	+	Create
2-Saddle	-	Cut
2-Saddle	+	Attach
1-Saddle	-	Pierce
1-Saddle	+	Spackle
Minimum	-	Bubble
Minimum	+	Burst

Table 2: The affect of critical point sign on topology.

### 4.2 Identifying Polygons to Remove

Changes in maximum and minimum critical values cause entire simply-connected components of polygonization to be removed or created. Changes in saddle points require the determination of specific polygons to be removed such that their vertices may be properly reconnected. These polygons intersect a separatrix extending from the saddle point.

The separatrix may be efficiently approximated by a line for 2-saddles, or a plane for 1-saddles. These lines and planes described by the eigenvectors of the stability matrix of a critical point approximate the separatrix.

When separatrices are linearly approximated by lines and planes, certain errors might occur. For example, a 2-saddle may connect two components, but the line approximating its separatrix might not intersect either component. One must then assume that the parameter vector  $\mathbf{q}$  is sufficiently close to the parameter vector at the topology change  $\mathbf{q}_*$  that the linear approximation correctly intersect the proper polygonized implicit surface components.

The separatrix extending from a 2-saddle can be treated as an initial value problem, using the positive eigenvector  $\mathbf{v}_3(\mathbf{x})$  of the stability matrix to define the ordinary differential equation

$$\dot{\mathbf{x}} = \mathbf{v}_3(\mathbf{x}) \quad (6)$$

and using numerical integration to trace out the path of the separatrix. The midpoint method provided sufficient numerical accuracy for this task in our experiments.

The case where a separatrix intersects a polygonization vertex can be removed with a topology-preserving perturbation.

### 4.3 Reconnection

The following procedures describe which polygons must be removed, and how their vertices are reconnected to update the topology of the polygonization.

**Destroy.** When the value at a maximum goes negative, an isolated component in the implicit surface disappears. A ray cast from the maximum point in any direction will first intersect a polygon in this simply-connected component. All polygons connected to this polygon are then removed. Figure 7 pseudocodes this algorithm.

**Create.** When the value at a maximum goes positive, a new, simply-connected component in the implicit surface appears. A sufficiently small tetrahedron may be placed around the maximum point, letting its vertices adhere to the implicit surface component and adding

```

Procedure Destroy/Burst
  Cast a ray from maximum point in any direction.
  Let  $p$  be the first polygon the ray intersects.
  Push  $p$  onto stack.
  While stack not empty.
    Let  $p$  be the result of popping the stack.
    For all polygons  $q$  sharing an edge with  $p$ .
      Push  $q$  onto stack.
  Remove  $p$  from polygonization.

```

Figure 7: The repolygonization algorithm for *Destroy* and *Burst*.

new polygons when necessary. Alternatively, a ray may be cast from the maximum point and intersected with the implicit surface. The first intersection denotes the location where any standard continuation polygonization technique may be applied. Figure 8 pseudocodes this algorithm.

```

Procedure Create/Bubble
  Cast a ray from maximum point in any direction.
  Let  $\mathbf{x} \in f^{-1}(0)$  be the first ray intersection.
  Polygonize the component containing  $\mathbf{x}$ .

```

Figure 8: The repolygonization algorithm for *Create* and *Bubble*.

**Cut.** When the value at a 2-saddle goes negative, part of the implicit surface disconnects. The separatrix surface extending from the 2-saddle is found by integrating the two negative eigenvalues of the stability matrix will intersect the polygons in a ring surrounding the 2-saddle. In practical cases, this separatrix is sufficiently approximated by a plane passing through the 2-saddle perpendicular to its positive eigenvector. The ring of polygons intersecting the separatrix surface are removed, yielding two disjoint rings of polygonization vertices. These rings are “sewn up” individually via triangulation. Figure 9 pseudocodes this algorithm, and Figure 10 illustrates the polygon configuration.

```

Procedure Cut/Spackle
  Let  $P$  be a plane containing the critical point  $\mathbf{x}$  perpendicular to the uniquely-signed eigenvector of  $V(\mathbf{x})$ .
  Cast a ray from  $\mathbf{x}$  in any direction within  $P$ .
  Let  $p_0$  be the first polygon intersected by the ray.
  Initialize  $i = 0$  and repeat.
    Let  $p_{i+1}$  be a polygon intersecting  $P$ , sharing an edge with  $p_i$ , and not equal to any  $p_j$  for  $j \leq i$ .
    If no such  $p_{i+1}$  exists, break.
    Increment  $i$ .
  Let  $v_0$  be any vertex of  $p_0$ .
  Call Procedure Ring.
  Triangulate  $v_i$ .
  Let  $v_0$  be any vertex of  $p_0$  not currently triangulated.
  Call Procedure Ring.
  Triangulate  $v_i$ .

Procedure Ring
  Initialize  $i = 0$  and repeat.
    Let  $e$  be an edge connecting vertex  $v_i$  to  $v_{i+1}$  separating a  $p_k$  polygon from a non- $p_k$  polygon, and vertex  $v_{i+1}$  not equal to any  $v_j$  for  $j \leq i$ .
    If no such  $v_{i+1}$  exists, break.
    Increment  $i$ .

```

Figure 9: The repolygonization algorithm for *Cut* and *Spackle*.

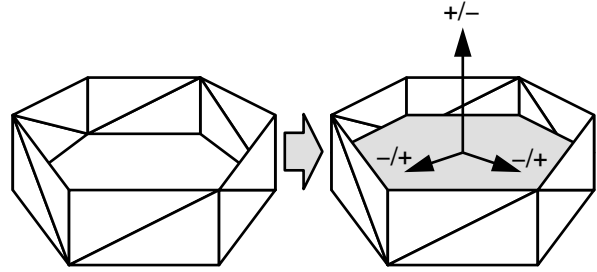
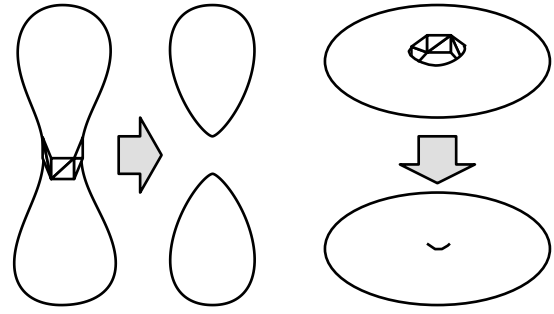


Figure 10: Polygons, eigenvalues and eigenvectors for *Cut* and *Spackle*.

**Attach.** When the value at a 2-saddle goes positive, two components of the implicit surface connect. The separatrix curve extends from the 2-saddle in the direction of its positive eigenvalue to a maximum point inside each component. The first polygon in each direction the separatrix intersects is removed. This leaves two disjoint rings of vertices that need to be connected. Proper correspondence algorithms between the two polygons can be found (e.g. [26]), but such techniques are not necessary if the polygonization is restricted to triangles. Figure 11 pseudocodes this algorithm.

```

Procedure Attach/Pierce
  Extend separatrix curves from the critical point.
  Let polygons  $p_0$  and  $p_1$  first intersect each separatrix.
  Connect the vertices of  $p_0$  with the vertices of  $p_1$ .
  Remove  $p_0$  and  $p_1$ .

```

Figure 11: The repolygonization algorithm for *Attach* and *Pierce*.

**Pierce.** When the value at a 1-saddle goes negative, a hole is pierced in the implicit surface. Similar to the *attach* case (a hole in the implicit surface of  $f$  is a connection in the implicit surface of  $-f$ ), the two polygons that intersect the separatrix curve passing through the 1-saddle in the direction of the eigenvector corresponding to the one negative eigenvalue of the stability matrix are identified. These two polygons are removed and now form the ends of the hole. Corresponding and connecting the resulting two rings of vertices form the walls of the hole. The algorithm in Figure 11 also repolygonizes the *pierce* case.

**Spackle.** When the value at a 1-saddle goes positive, a hole in the implicit surface is filled. Similar to the *cut* case, the separatrix surface is constructed at the 1-saddle perpendicular to the eigenvector corresponding to the one negative eigenvalue of the stability matrix. The local polygons this surface pierces are removed, and the two resulting polygonal holes are “sewn up” by triangulation. The algorithm in Figure 9 also repolygonizes the *spackle* case and Figure 10 illustrates the polygon configuration.

**Bubble.** When the value at a minimum goes negative, a pocket of

air forms inside the implicit solid. An air pocket in the implicit surface of  $f$  is a new component in the implicit surface of  $-f$ . This pocket of air may therefore be treated as a simply-connected implicit surface component, and polygonized using any of the existing techniques. The algorithm in Figure 8 also repolygonizes the *bubble* case.

**Burst.** When the value at a minimum goes positive, an air bubble within the implicit solid has burst. As in the Destroy case, a ray is cast from the minimum in any direction. The first polygon this ray intersects, as well as any other polygons with a connection to it, are then removed. The algorithm in Figure 7 also repolygonizes the *burst* case.

## 5 Polygonization

Morse theory provides the background for a topologically-guaranteed polygonization algorithm. Given a function  $f(\mathbf{x})$  implicitly defining the surface  $f^{-1}(0)$ , we consider the family of surfaces  $f^{-1}(a)$  for non-negative  $a$ . Let  $a_0$  be a value of sufficient magnitude such that the surface  $f^{-1}(a_0) = \emptyset$ . As  $a_0$  decreases, it will pass critical values for maximum points, 2-saddles, 1-saddles and minimum points. As each critical value is encountered, the topology of the polygonization around its critical point is corrected. This “inflation” algorithm is pseudocoded in Figure 12.

```

Procedure Inflate
   $X_c = \text{Search } \mathbf{X} \text{ for } \{\mathbf{x} : \nabla f(\mathbf{x}) = 0\}.$ 
  Let  $a_0 > \max_{\mathbf{x} \in X_c} f(\mathbf{x})$ .
  Polygonize  $f^{-1}(a_0)$ .
  For  $a = a_0 - \epsilon$  to 0 step  $-\epsilon$ .
    Adjust vertices to  $f^{-1}(a)$ .
    If  $\exists \mathbf{x} \in X_c : a < f(\mathbf{x}) < a + \epsilon$  then
      Correct topology change in polygonization.
  Return polygonization of  $f^{-1}(0)$ .

```

Figure 12: The “Inflate” polygonization algorithm.

When a maximum point is passed, a new simply-connected component appears via the *create* routine. When a 2-saddle is passed, a connection formed via the *attach* routine. When a 1-saddle is passed, a hole is filled via the *spackle* routine. When a minimum point is passed, a hollow bubble is filled via the *burst* routine. The *Inflation* polygonization of a blobby cube is demonstrated in Figure 13

Shrinkwrapping similarly polygonizes implicit surfaces but from the opposite direction, approaching the isovalue from the negative side [34]. Hence, the polygonization begins with a large simply-connected spheroid, which shrinks and appears to adhere to the final implicit surface. Morse theory can be incorporated to detect changes in topology during the shrinkwrapping process [4].

One problem with shrinkwrapping is that its outside-in processing fails to account for hollow bubbles within an implicit surface whereas *inflate*’s inside-out processing correctly detects and polygonizes these regions. While such regions are typically hidden from the viewer, they become visible when the surface is rendered translucently or when the surface’s polygonization is later intersected, trimmed, clipped or blended.

## 6 Interactive Repolygonization

The interaction algorithm consists of an initialization stage followed by an interactive loop of user input, model update, and model display. The system assumes it is initialized with a topologically correct polygonization, such as is described in Section 5.

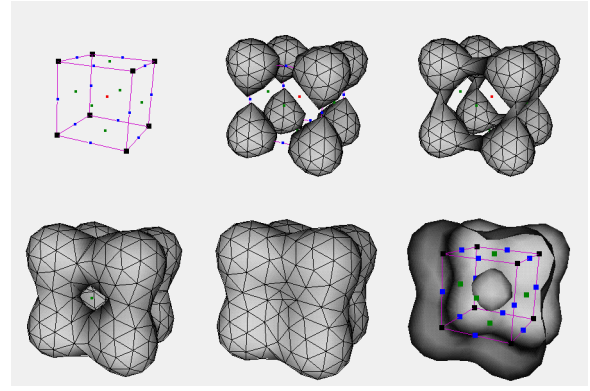


Figure 13: Polygonization via *Inflation* of the blobby cube. The last image illustrates the air bubble by only rendering back-facing polygons. Note that the definition of back facing may be the opposite of one’s intuition for an air bubble.

```

Procedure Shrinkwrap
   $X_c = \text{Search } \mathbf{X} \text{ for } \{\mathbf{x} : \nabla f(\mathbf{x}) = 0\}.$ 
  Let  $a_0 < \min_{\mathbf{x} \in X_c} f(\mathbf{x})$ .
  Polygonize  $f^{-1}(a_0)$ .
  For  $a = a_0 + \epsilon$  to 0 step  $\epsilon$ .
    Adjust vertices to  $f^{-1}(a)$ .
    If  $\exists \mathbf{x} \in X_c : a < f(\mathbf{x}) < a + \epsilon$  then
      Correct topology change in polygonization.
  Return polygonization of  $f^{-1}(0)$ .

```

Figure 14: The “Shrinkwrap” polygonization algorithm [4].

For each time step, the interaction algorithm performs the steps in Figure 15.

```

Procedure InteractionLoop
  Repeat.
    Alter model parameters based on user manipulation.
    Adjust vertex positions, fix mesh.
    Determine critical points.
    Correct polygonization topology.
  Render.

```

Figure 15: The interaction loop for interactive implicit surface modeling.

Figure 16 shows a critical point tracking algorithm. During user interaction, the critical points move and change sign. Furthermore, one or more of the eigenvalues of the stability matrix can change sign at some degenerate critical point  $\mathbf{x}$ , resulting in the creation or annihilation of a pair of critical points. Critical point annihilation is revealed by the collision of two critical-point tracking particles. Critical point creation occurs spontaneously and is not detected by the tracking particles.

Searching in four-dimensions, as shown in Figure 17, allows us to find the location in space and time of degenerate critical points. Since critical points can be tracked and annihilation can be detected, the interval search would serve to detect the spontaneous creation of critical points. Such occurrences rarely happen, but when they do occur they appear as double zeros (both  $\nabla f(\mathbf{x})$  and  $|V(\mathbf{x})| = 0$ ) which degrades convergence.

An alternative search shown in Figure 18 finds the location in space and time for singular points in the family of implicit surfaces, where the value of a critical point changes sign. Such occurrences

```

Procedure Track-n-Search
  Track critical points.
  Search  $\mathbf{X}$  for  $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}\}$ .

```

Figure 16: The “Track-n-Search” critical point determination algorithm.

```

Procedure Track-n-SearchDegenerate
  Track critical points.
  Search  $\mathbf{X} \times [t_0, t_1]$  for  $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}, |V(\mathbf{x})| = 0\}$ .

```

Figure 17: The “Track-n-SearchDegenerate” critical point determination algorithm.

occur as rarely as degenerate critical points, so the interval search can quickly guarantee that such points do not exist. However, when they do exist, as with degenerate critical points, they appear as double zeros which degrades the rate of convergence.

## 7 Conclusion

Using techniques from catastrophe theory and Morse theory, the preceding sections developed (1) a new polygonization algorithm that can guarantee the topology of the polygonization matches that of the implicit surface, and (2) a new implicit surface modeling system capable of maintaining a topologically-accurate polygonized representation of the implicit surface during direct manipulation at interactive update rates.

Section 4 improves previous *ad hoc* geometry-only techniques [24, 4] by describing a mathematically sound method for using the separatrix to identify the polygons affected by a topology change, and robust algorithms for reconnecting the vertices of the polygonization.

Section 5 uses these techniques to polygonize an implicit surface. This method improves previous geometry-based interval methods [28] in that it is faster and does not return a large number of unnecessarily small polygons. The interval search is also guaranteed to find all critical points, which overcomes the uncertainty of previous methods [4], and also properly polygonizes hollow bubbles when they appear within an implicit surface.

Some initial experiments revealed that performance dropped below ten frames per second on scenes containing combinations of four or more interacting blobby ellipsoids. Modeling sessions that string a chain of blobby components operate in real time, but sessions with densely packed arrangements of blobby components appear sluggish in the current prototype implementation of the system. Even apparently simple configurations of blobby components can yield numerous nearly-degenerate critical points, and their detection is required for accurate topology management. This performance was measured using the SearchSingularity interaction loop, but is similar to the performance of the other interaction loop critical point search/tracking methods. This procedure becomes noticeably slow near topology changes. While any speed degradation and inconsistency is undesirable, the algorithm does focus its computation on the time and space where it is most needed.

```

Procedure SearchSingularity
  Search  $\mathbf{X} \times [t_0, t_1]$  for  $\{\mathbf{x} : f(\mathbf{x}) = 0, \nabla f(\mathbf{x}) = \mathbf{0}\}$ .

```

Figure 18: The “SearchSingularity” algorithm.

## 7.1 Some Implementation Tricks

One of the topological guarantee’s restrictions was the lack of degenerate critical points. However, for speed, we were able to implement a  $C^2$  cubic approximation to the exponential blobby model. The kernel of this approximation is uniform away from the center, and results in a three-dimensional degenerate critical set. We overcame this problem by assuming that the derivative intervals with zero for one endpoint did not contain a critical point, but instead contained a portion of this 3-D critical set. We avoided the possibility that a critical point fell on the boundary of the interval by expanding each interval by a small percentage.

Occasionally, the program errs in its attempt to process a topology change. In such cases, the system automatically initiates a full “inflation” repolygonization.

Further implementation details can be found in the dissertation [29].

## 7.2 Future Work

Tracking critical points is much faster than searching for them, but does not account for the pairs of critical points that can be created spontaneously. Tracking all of the derivatives of the function could detect degenerate critical points. This is not possible for exponentials because they are infinitely differentiable, and their derivatives become increasingly complex. Piecewise polynomials have finitely many derivatives that become increasingly simple, and might offer the opportunity to attempt such tracking.

Implicit surfaces still offer many challenges in modeling, texturing and animation due to the flexibility of their topology. This research solved the problem of interactive polygonization. Understanding the dynamics of critical points might lead to further solutions to other implicit surface problems, such as maintaining a consistent texturing during a topology change.

This research focused on 3-D implicit surfaces. Its application to the polygonization and modeling of 2-D implicit curves would be a useful, though perhaps now trivial, simplification.

## 7.3 Acknowledgments

This research was supported in part by the NSF Research Initiation Award #CCR-9309210. This research was performed in the Imaging Research Laboratory. The authors would like to thank the SIGGRAPH reviewers for their constructive criticism and positive comments. Further thanks are due to Dan Asimov, Jules Bloomenthal and Jim Kajiya for their help in tracking down theorems in Morse theory. Special thanks to Andrew Glassner and Scott Lang for their assistance with the photoready copy of this paper.

## REFERENCES

- [1] BLINN, J. F. A generalization of algebraic surface drawing. *ACM Transactions on Graphics* 1, 3 (July 1982), 235–256.
- [2] BLOOMENTHAL, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5, 4 (Nov. 1988), 341–355.
- [3] BLOOMENTHAL, J., AND WYVILL, B. Interactive techniques for implicit modeling. *Computer Graphics* 24, 2 (Mar. 1990), 109–116.
- [4] BOTTINO, A., NUIJ, W., AND VAN OVERVELD, K. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 53–72.



- [5] CHENG, K.-P. Using plane vector fields to obtain all the intersection curves of two general surfaces. In *Theory and Practice of Geometric Modeling* (New York, 1989), Springer-Verlag.
- [6] DE FIGUEIREDO, L. H., DE MIRANDA GOMES, J., TERZOPOULOS, D., AND VELHO, L. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92* (May 1992), pp. 250–257.
- [7] DELMARCELLE, T., AND HESSELINK, L. The topology of symmetric, second-order tensor fields. *Proceedings IEEE Visualization '94* (October 1994), 140–147.
- [8] DESBRUN, M., TSINGOS, N., AND GASCUEL, M.-P. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Implicit Surfaces '95 Proceedings* (April 1995), 171–185.
- [9] FLEISCHER, K. W., LAIDLAW, D. H., CURRIN, B. L., AND BARR, A. H. Cellular texture generation. In *Computer Graphics (Annual Conference Series)* (Aug. 1995), pp. 239–248.
- [10] HANSEN, E. A globally convergent interval method for computing and bounding real roots. *BIT* 18 (1978), 415–424.
- [11] HANSEN, E. R., AND GREENBERG, R. I. An interval newton method. *Applied Mathematics and Computation* 12 (1983), 89–98.
- [12] HART, J. C. Morse theory for computer graphics. Tech. Rep. EECS-97-002, Washington State University, May 1997. Also in: SIGGRAPH '97 Course #14 Notes “New Frontiers in Modeling and Texturing”.
- [13] HELMAN, J. L., AND HESSELINK, L. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications* (May 1991), 36–46.
- [14] KALRA, D., AND BARR, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics* 23, 3 (July 1989), 297–306.
- [15] KERGOSIEN, Y. L. Generic sign systems in medical imaging. *IEEE Computer Graphics and Applications* 11, 5 (Sep. 1991), 46–65.
- [16] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics* 21, 4 (July 1987), 163–170.
- [17] MITCHELL, D. Three applications of interval analysis in computer graphics. In *Frontiers of Rendering*. SIGGRAPH '91 Course Notes, 1991.
- [18] MITCHELL, D., AND HANRAHAN, P. Illumination from curved reflectors. *Computer Graphics* 26, 2 (July 1992), 283–291.
- [19] MOORE, R. E. *Interval Analysis*. Prentice Hall, 1966.
- [20] NING, P., AND BLOOMENTHAL, J. An evaluation of implicit surface tilers. *Computer Graphics and Applications* 13, 6 (Nov. 1993), 33–41.
- [21] NISHIMURA, H., HIRAI, M., KAWAI, T., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. Object modeling by distribution function and a method of image generation. In *Proc. of Electronics Communication Conference '85* (1985), pp. 718–725. (Japanese).
- [22] NORTON, A. Generation and rendering of geometric fractals in 3-D. *Computer Graphics* 16, 3 (1982), 61–67.
- [23] RATSCHKE, H., AND ROKNE, J. *Computer Methods for the Range of Functions*. John Wiley and Sons, 1984.
- [24] RODRIAN, H.-C., AND MOOCK, H. Dynamic triangulation of animated skeleton-based implicit surfaces. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 37–52.
- [25] ROSCH, A., RUHL, M., AND SAUPE, D. Interactive visualization of implicit surfaces with singularities. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 73–87.
- [26] SEDERBERG, T. W., AND GREENWOOD, E. A physically based approach to 2-D shape blending. *Computer Graphics* 26, 2 (July 1992), 25–34.
- [27] SHINAGAWA, Y., KUNII, T. L., AND KERGOSIEN, Y. L. Surface coding based on morse theory. *IEEE Computer Graphics and Applications* 11, 5 (Sep. 1991), 66–78.
- [28] SNYDER, J. *Generative Modeling for Computer Graphics and CAD*. Academic Press, 1992.
- [29] STANDER, B. T. *Polygonizing Implicit Surfaces with Guaranteed Topology*. PhD thesis, School of EECS, Washington State University, May 1997.
- [30] SUFFERN, K., AND FACKERELL, E. Interval methods in computer graphics. In *Proc. AUSGRAPH 90* (1990), pp. 35–44.
- [31] SZELISKI, R., AND TONNESEN, D. Surface modeling with oriented particle systems. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 185–194.
- [32] TAYLOR, A. E. *Advanced Calculus*. Ginn and Company, 1955.
- [33] TURK, G. Generating textures for arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (July 1991), T. W. Sederberg, Ed., vol. 25, pp. 289–298.
- [34] VAN OVERVELD, C., AND WYVILL, B. Shrinkwrap: an adaptive algorithm for polygonizing and implicit surface. Tech. Rep. 93/514/19, University of Calgary, Dept. of Computer Science, March 1993.
- [35] WITKIN, A. P., AND HECKBERT, P. S. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)* (July 1994), pp. 269–278.
- [36] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *Visual Computer* 2, 4 (1986), 227–234.

# Using the CW-Complex to Represent the Topological Structure of Implicit Surfaces and Solids.

John C. Hart  
 School of EECS  
 Washington State University  
 Pullman WA 99164-2752  
 (509) 335-2343  
 hart@eecs.wsu.edu

## Abstract

We investigate the CW-complex as a data structure for visualizing and controlling the topology of implicit surfaces. Previous methods for controlling the blending of implicit surfaces redefined the contribution of a metaball or unioned blended components. Morse theory provides new insight into the topology of the surface a function implicitly defines by studying the critical points of the function. These critical points are organized by a separatrix structure into a CW-complex. This CW-complex forms a topological skeleton of the object, indicating connectedness and the possibility of connectedness at various locations in the surface model. Definitions, algorithms and applications for the CW-complex of an implicit surface and the solid it bounds are given as a preliminary step toward direct control of the topology of an implicit surface.

## 1 Introduction

The holy grail of implicit surface modeling research is the control of blending. Most implicit surface primitives blend based on nothing more than proximity, often blending in undesirable locations. Current solutions have created new implicit primitives that do not blend, or use blending graphics that can create unsightly creasing.

Blending usually changes the topology of the surface, so we turn to tools from topology to analyze the unwanted blending problem. Morse theory shows how the topology of a surface implicitly defined by a function can be discerned by the arrangement of function's critical points. A theorem from Morse theory describes a structure that represents the topology of the surface called the CW-complex. We therefore propose using the CW-complex as a data structure for representing the topological structure of implicit surfaces. We present algorithms for computing the embedding of the CW-complex on an implicit surface and within an implicit solid.

## 2 The Unwanted Blending Problem

The implicit surfaces most in need of attention to topology is the metaball or blobby model. This model consists of spheres blended by adding quasi-Gaussian three-dimensional bump functions centered at key points in space. As the blending

of these metaballs depends primarily on proximity, unwanted blending often occurs. For example, a hand can be constructed from metaballs by blending fingers with a palm. However, if the tips of any two fingers get too close to each other, then they too will blend together.

Gascuel [1993] showed how to simulate *precise contact* between implicit surfaces. (The following summary is a gross simplification, and, for example, omits all details of physical modeling and propagation regions.) If the implicit surface of  $f_i$  comes into contact with the surface of  $f_j$ , then the resulting deformation on  $f_i$  is given by

$$g_i(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}). \quad (1)$$

The metaball of  $f_j$  has been replaced with a negative metaball of  $-f_j$  to determine the effect on metaball of  $f_i$ . A similar range deformation occurs on metaball  $j$ . The resulting implicit surfaces of  $g_i$  and  $g_j$  may touch but can not interpenetrate, for if any point  $\mathbf{x}$  we have  $g_i(\mathbf{x}) > 0$  then  $f_i(\mathbf{x}) > f_j(\mathbf{x})$  and therefore  $g_j < 0$ . Precise contact modeling provides a method for preventing blending and even interpenetration between two proximate metaballs, but can deform the metaballs even if they do not intersect.

Guy & Wyvill [1996] proposed a blending graph to control the blending of metaballs. (The following summary focuses only on controlling the blending, and omits the precise contact component.) The blending graph  $G$  is a graph that contains a vertex for every metaball, and an edge  $(i, j)$  denoting that metaballs  $i$  and  $j$  should blend together. Hence for each metaball function  $f_i$ , a new function is defined as

$$g_i(\mathbf{x}) = f_i(\mathbf{x}) + \sum_{(i,j) \in G} f_j(\mathbf{x}). \quad (2)$$

The resulting implicit surface is then the union (CSG, not blended) of the implicit surfaces of the  $g_i$

$$G(\mathbf{x}) = \max_i g_i(\mathbf{x}). \quad (3)$$

This union operation results in creases in the implicit surfaces due to the  $C^1$  discontinuities introduced by the *max* operation. These discontinuities could be removed by replace the maximum with an *R-function* representation [Pasko *et al.*, 1995] but this could reintroduce unwanted blending.

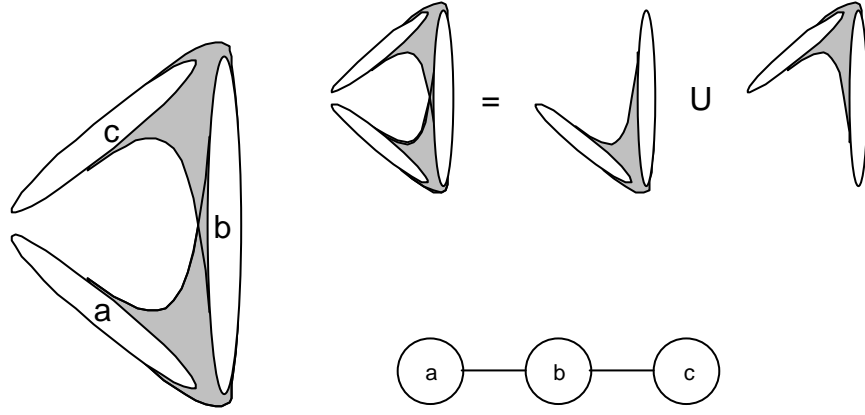


Figure 1: The blend graph controlling the blending of three meta-ellipses.

### 3 Morse Theory

The blending of metaballs, intentional or not, is a change in the topological type of the shape. Hence we use techniques from topology to analyze the problem of unwanted blending. We base our analysis on Morse theory, which describes the topology of surfaces by the configuration of critical points, and use a data structure from algebraic topology called the CW-complex to represent the relationship of critical points and the topological structure of the surface.

We assume surfaces are defined in *general position*, such that form of the surface remains unchanged after a perturbation of the surface's parameters. The assumption of general form means that there may be special cases in which this paper's claims do not hold and its algorithms malfunction, but such cases are isolated in the parameter space of the surface and may be removed by perturbing these parameters.

Let  $f$  be a smooth real function on a smooth manifold  $M^1$ . The *critical points* of  $f$  are the points  $p \in M$  where its gradient  $\nabla f(p)$  vanishes. The *critical value* is the value of  $f$  at a given critical point.

The Hessian  $V(p)$  of  $f$  at  $p \in M$  yields a square symmetric matrix of second derivatives of  $f$ . Let  $\lambda_i$  denote the  $i$ th eigenvalue, in order of non-decreasing value, and let  $\mathbf{v}_i$  denote its corresponding eigenvector. Critical points are classified by the number of negative eigenvalues of this matrix. The function  $f$  is called a *Morse function* if the Hessian is nonsingular for every  $p \in M$ . If any of these eigenvalues are zero, then the critical point is *degenerate*. We assume the surface model is sufficiently parameterized such that these degeneracies are special cases and can be removed via perturbation.

Morse theory provides a connection between the arrangement of these critical points and the topology of  $M$  [Milnor, 1963]. For example, let 0 be a regular value of  $g : \mathbb{R}^3 \rightarrow \mathbb{R}$ . Then by the implicit function theorem, its inverse image  $g^{-1}(0)$  is a manifold and is called the implicit surface of  $g$ .

<sup>1</sup>Note that with special care, Morse theory can be applied to manifolds of continuity as low as  $C^0$  [Goresky & MacPherson, 1988] and for functions of continuity as low as  $C^1$  [Hart *et al.*, 1998].

Let  $f$  be a *height function* on  $g^{-1}(0)$  such that  $f(x, y, z) = y$ . Then  $f$  is a Morse function. Using a classic example of Bott, let  $g^{-1}(0)$  be a torus encircling the  $z$ -axis. Then there exist four critical points such that

$$\nabla f = (0, 1, 0) \cdot \nabla g = \partial g / \partial y \quad (4)$$

vanishes: one minimum at the bottom of the torus, one maximum at the top of the torus, and two saddle points on the top and bottom of the hole. If we were to construct the torus from the bottom up, we see that a component of the surface was created at the minima, opposing sections of the surface joined at the saddles, and the surface closed at the top.

If the manifold is  $\mathbb{R}^3$ , then Morse theory can be used to define the topology of the implicit solid of  $f$  [Hart, 1998; Fomenko & Kunii, 1997]. Critical points are classified in four categories: maxima (3-saddles), 2-saddles, 1-saddles and minima (0-saddles)<sup>2</sup>. For implicit surfaces constructed with metaballs, the critical points occur at key locations defining the topology of the surface. A maxima occurs near the center of a metaball component<sup>3</sup>. A component is created if this critical value is positive, and is destroyed if it is negative. A 2-saddle occurs between pairs of metaballs. These metaballs are connected if this critical value is positive, and are disconnected if it is negative. A 1-saddle occurs in the middle of a ring of three or more metaballs. This ring is filled if this critical value is positive, and is pierced if it is negative. A minimum occurs inside a non-planar collection of four or more metaballs. The collection solid if this critical value is positive, but contains an air bubble if the critical value is negative.

### 4 The CW-Complex

In algebraic topology, specifically homotopy theory, shapes are often constructed out of cells of different dimension. An

<sup>2</sup>Johnson *et al.* [1999] called the critical points of real functions on  $\mathbb{R}^3$  *peaks, passes, pales and pits*.

<sup>3</sup>Stander & Hart [1997] provides an example where ellipsoidal metaballs can blend to yield an additional disjoint "phantom" component.

$n$ -cell is denoted  $e^n$  and represents a building block of dimension  $n$ . For example, a 0-cell is a point, a 1-cell is a space curve segment, a 2-cell is a surface patch and a 3-cell is a solid. The  $n$ -cells are closed such that the space curve segment includes its endpoints, the patch includes its boundary curve and the solid includes its boundary surface.

Each  $n$ -cell is homeomorphic to the  $n$ -ball

$$B^n = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leq 1\}. \quad (5)$$

This homeomorphism also maps the *boundary of a  $n$ -cell* to the  $(n - 1)$ -sphere defined

$$S^n = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = 1\}. \quad (6)$$

It is helpful to note that  $S^{-1} = \emptyset$  and this boundary (from algebraic topology) is not the same as the boundary from point-set topology (closure minus interior).

We can now construct a *CW-complex* out of  $n$ -cells of increasing dimensions. An  $n$ -cell is attached to a CW-complex by *identifying* the boundary of the cell with the union of some collection of  $(n-1)$ -cells in the complex.

Hence, we can construct a 3-dimensional CW-complex. We begin with the empty set. We then attach 0-cells by unioning disjoint points into the set. We attach 1-cells by unioning space curve segments whose endpoints lie on these points. We attach 2-cells by unioning surface patches whose boundaries lie on the space curve segments. We attach 3-cells by filling in closed regions bounded by space curves.

Note that the CW-complex generalizes the notion of a graph by adding cells of dimension greater than 1. Also note that a CW-complex generalizes the notion of a simplicial complex in that triangles become "polygons" and tetrahedra become "polyhedra." Note also that the cells need not be straight, such that a "polygon" may contain as few as a single edge connecting a single vertex to itself, and a "polyhedron" may contain as few as a single face, a single vertex and no "edges."

A theorem [Milnor, 1963] from classical Morse theory states that a compact manifold has the homotopy type of a CW-complex consisting of a  $\lambda$ -cell for each critical point of type  $\lambda$  for a given Morse function. The proof of this theorem is based on the analogous operations of attaching a handle to a manifold and attaching a  $\lambda$ -cell to a CW-complex. The remainder of this paper describes how this CW-complex may be instead constructed from the separatrix structure connecting the critical points to each other.

## 5 The CW-Complex of an Implicit Surface

As before let  $g$  implicitly define the surface  $g^{-1}(0)$  and let  $f$  be a Morse function on the surface. Let  $\mathbf{x}_i$  be the critical points of  $f$  on  $g^{-1}(0)$ , and let  $c_i = f(\mathbf{x}_i)$  be the corresponding critical values.

The 0-cells of the CW-complex exist at the minima of the surface. The 1-cells are separatrix curves constructed by integrating the ordinary differential equation

$$\dot{\mathbf{x}} = \pm \mathbf{v}_0(\mathbf{x}) \quad (7)$$

with the initial values set to the saddle points, and termination values set to the minima. Situations where this curve leads

from a saddle to another saddle are unstable and removed via perturbation.

The 2-cells are the remaining components of the implicit surface, and can be interrogated using, for example, surface particles [Witkin & Heckbert, 1994]. Each of these remaining components will contain a single maximum which can be used as the seed point for the birth of a surface particle population. These surface particles can be further constrained to never cross a separatrix curve.

Figure 2 demonstrates the procedure on a torus. Note that if the surface contains a single minimum, the resulting CW-complex is a "bouquet" and is easily "unfolded" into its fundamental polygon [Fomenko & Kunii, 1997]. This form of the CW-complex provides a representation of the topology of the surface embedded on the surface itself.

## 6 The CW-Complex of an Implicit Solid

Likewise, one may compute the CW-complex of an implicit solid.

0-cells of the CW-complex are placed on the maxima. Separatrix curves are traced through each 2-saddle by integrating the ordinary differential equation

$$\dot{\mathbf{x}} = \pm \mathbf{v}_2(\mathbf{x}) \quad (8)$$

until they reach the maxima. Note that these separatrix curves can not stably reach any critical point other than a maxima.

We again use surface particles to interrogate the 2-cells of the CW-complex. The surface particles are spawned from the 1-saddles, and remain on the separatrix surface by the constraint

$$\dot{\mathbf{x}} \cdot \mathbf{v}_0(\mathbf{x}) = 0. \quad (9)$$

The surface particles are further constrained to not cross the boundaries of the separatrix curves defined earlier.

The 3-cells are now well defined as the compact spatial regions bounded by the separatrix surfaces, and each 3-cell will contain a single minima critical point. These regions could be interrogated by any volumetric region growing method.

Note that we have reversed the dimension of the cells with respect to the index of the critical points (e.g. a 0-cell corresponds to an index 3 critical point — a maximum). This is an artifact of the sign convention of metaballs disagreeing with the sign convention of Morse theory. Note that the cell dimension would agree with the critical point index for the function  $-f$  which has an identical implicit surface, and also the same solid if the interior is defined for negative values instead of positive values.

Figure 3 demonstrates the CW-complex of a cuboid constructed from 8 metaballs, along with several implicit surfaces of the function for different isovalues. The critical structure includes 8 maxima at the vertices of the cube, 12 2-saddles centered along the edges of the cube, 6 1-saddles centered on the faces of the cube and a single minima at the centroid of the cube. The CW-complex thus contains 8 0-cells connected by 12 1-cells filled by 6 2-cells surrounding a single 3-cell.

The data structure described by this paper was also devised and used to represent crystal atomic structures [Johnson *et al.*, 1999; Johnson, 1999]. In fact, the Gaussian electron density

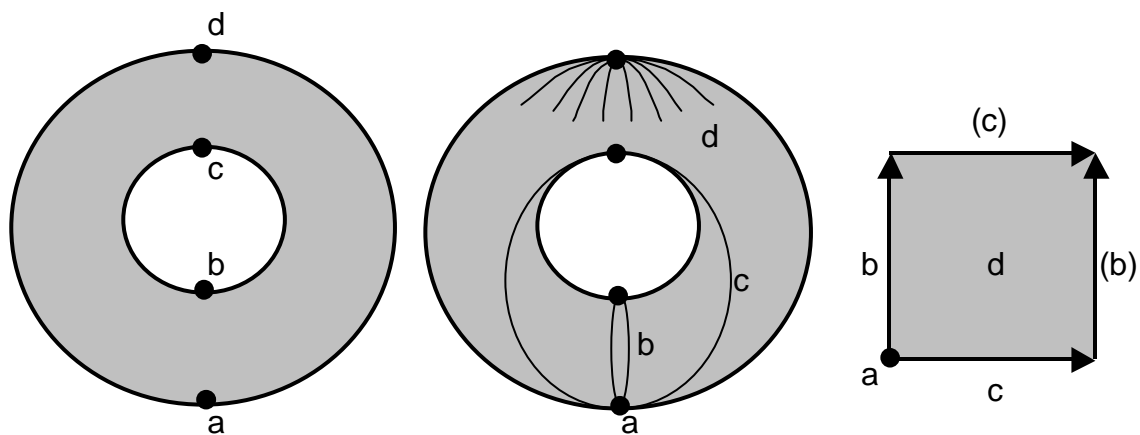


Figure 2: The CW-complex of a torus.

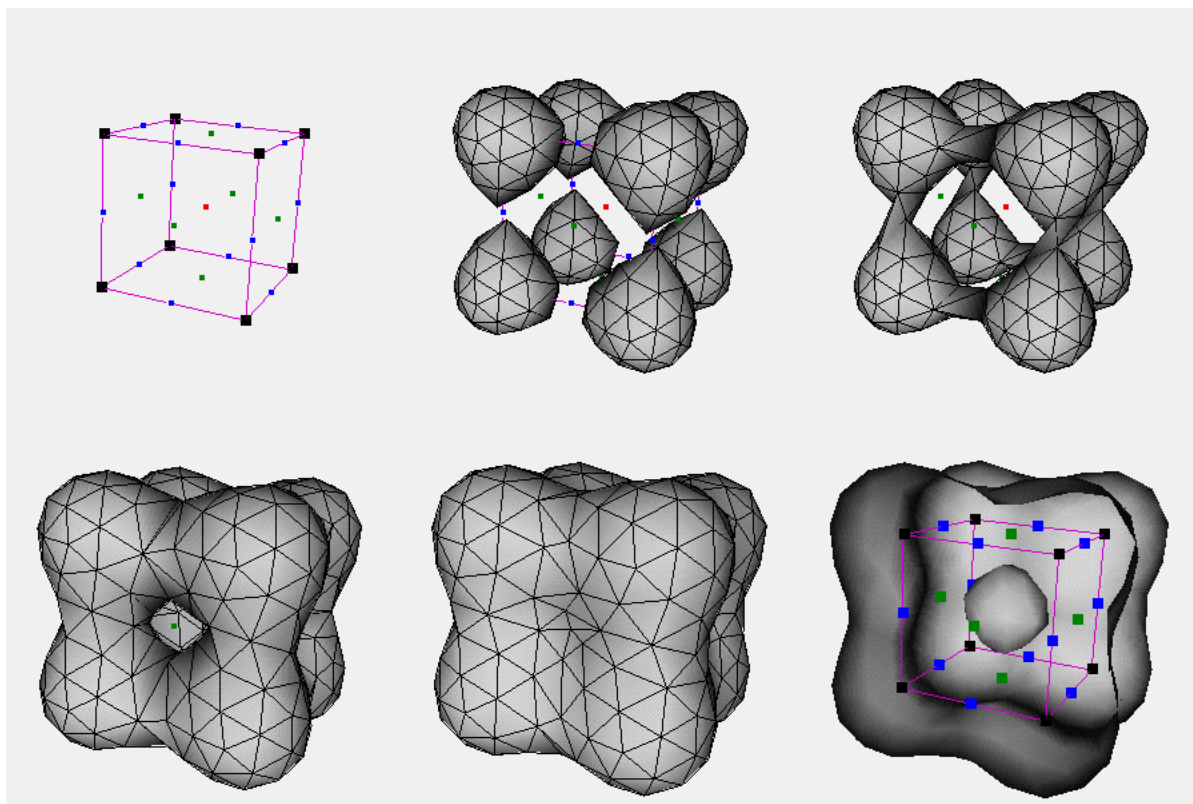


Figure 3: The CW-complex of a cuboid implicit solid.

maps used in crystallography are precisely the blobby objects we use in implicit surface modeling [Blinn, 1982]. The structure appearing in Figure 3 is the very same one [Johnson *et al.*, 1999] used to represent the topology of the crystalline structure of salt.

For implicit surfaces, the CW-complex corresponding to the defining function forms a skeletal structure. Whereas the medial-axis skeleton represents the geometry of the shape, the CW-complex skeletonizes the topology of the shape. Depending on the sign of the critical values, the components of the CW-complex can be drawn to better indicate their effect on the topology of the implicit surface. A 0-cell indicates a connected component surrounding a maxima, and can be represented by a small circle that is filled if the maxima's value is positive, or unfilled if it is negative. A 1-cell indicates a possible connection between components, and can be represented by a solid curve segment if the value of the corresponding 2-saddle is positive, or by a dashed segment if it is negative. A 2-cell indicates a region surrounded by a ring of metaballs, and can be represented by a hashing of solid lines if the value of the 1-saddle it contains is positive, or by a dotted surface if it is negative. A 3-cell indicates a region surrounded by a shell of metaballs, and can be represented by the patches on its boundary. These patches are brightly shaded if the value of the minima it surrounds is positive, or dimly shaded if it is negative.

We can replace the metaball kernel sum function  $f$  with a distance function

$$h(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) \min\{\|\mathbf{x} - \mathbf{y}\| \mid \mathbf{y} \in f^{-1}(0)\} \quad (10)$$

which shares the same sign and implicit surface as  $f$ . Then we hypothesize that the CW-complex constructed from the separatrix structure of this new function  $h$  is the medial-axis skeleton of the implicit surface. Note that this implies that the CW-complex of  $h$  never contains a minima, for it would then contain a 3-cell which would have been further "eroded" by the medial axis transform.

An alternative to the CW-complex for representing the critical structure is the *critical net* [Johnson *et al.*, 1999; Johnson, 1999]. The critical net is a simple graph representation of the separatrix structure of the critical points. This graph consists of a vertex for each critical point of any type. These vertices are connected by edges that indicate a separatrix connection between the critical points. A simple straight edge connects the separatrices between 2-saddles and maxima points. A separatrix curve can be extended from a 1-saddle to a 2-saddle by integrating

$$\dot{\mathbf{x}} = \pm \mathbf{v}_1(\mathbf{x}). \quad (11)$$

An edge replaces this curve connecting the vertices representing the 1-saddle and the 2-saddle. Similarly, a separatrix curve can be extended from the 1-saddle to the minimum by integrating

$$\dot{\mathbf{x}} = \pm \mathbf{v}_0(\mathbf{x}) \quad (12)$$

and replacing the separatrix curve by a straight edge between the vertices corresponding to the 1-saddle and the minimum.

## 7 Conclusion

The CW-complex of an implicit surface replaces the smooth surface with a quasi-polygonal version that can provide a sim-

pler representation of otherwise complex surfaces. In some cases, the CW-complex can be embedded in the plane providing further insight into the connectedness of the surface.

The CW-complex of an implicit solid displays the topology of the implicit surface to the user. Current controls would only allow the user to adjust the position, orientation or scale of metaballs. We envision a system that allows the user to directly control the topology of the implicit surface by application of range deformations that only change the sign of a specified critical point, but do not change the locations or number of the critical points.

The algorithms described focused on 2-manifold surfaces and 3-manifold solids. The techniques might also be helpful to understand the topology of higher dimensional manifolds. For example, in computer graphics, the bidirectional reflectance distribution function (BRDF) used for modeling local illumination is the 4-manifold defined as the explicit surface (graph) of the function  $\rho(\theta_i, \phi_i, \theta_r, \phi_r)$  with the identification  $\rho(\theta_i, \phi_i, \theta_r, \phi_r) = \rho(\theta_r, \phi_r, \theta_i, \phi_i)$ . A better understanding of the topology of this 4-manifold might lead to more efficient representations of it. Similarly, the plenoptic function used in image-based modeling and rendering also describes a 4-manifold whose topological understanding could lead to more efficient implementation.

This research was funded by a grant by the National Science Foundation. The author is grateful to fellow faculty members Ulrike Axen for discussions on the CW-complex and locating otherwise obscure references, and Bob Lewis for discussions on applications of computational topology to modeling illumination.

## References

- [Blinn, 1982] Blinn, J. F. A generalization of algebraic surface drawing. *ACM Transactions on Graphics* 1(3), July 1982, pp. 235–256.
- [Fomenko & Kunii, 1997] Fomenko, A. T. and Kunii, T. L. *Topological Modeling for Visualization*. Springer, 1997.
- [Gascuel, 1993] Gascuel, M.-P. An implicit formulation for precise contact modeling between flexible solids. In *Computer Graphics (Annual Conference Series.)*, Aug. 1993, pp. 313–320. Proc. SIGGRAPH 93.
- [Goresky & MacPherson, 1988] Goresky, M. and MacPherson, R. *Stratified Morse Theory*. Springer, April 1988.
- [Guy & Wyvill, 1996] Guy, A. and Wyvill, B. Controlled blending for implicit surfaces using a graph. In *Proc. Implicit Surfaces '95*. Eurographics, 1996, pp. 107–112.
- [Hart *et al.*, 1998] Hart, J., Durr, A., and Harsch, D. Critical points of polynomial metaballs. In *Proc. Workshop on Implicit Surfaces*. Eurographics/SIGGRAPH, June 1998, pp. 69–76.
- [Hart, 1998] Hart, J. C. Morse theory for implicit surface modeling. In Hege, H.-C. and Polthier, K., eds., *Mathematical Visualization*, pp. 257–268. Springer-Verlag, Heidelberg, 1998.

- [Johnson *et al.*, 1999] Johnson, C., Burnett, M., and Dunbar, W. Crystallographic topology and its applications. In Bourne, P. and Watenpaugh, K., eds., *Crystallographics Computing 7: Proceedings from the Macromolecular Crystallography Computing School*. University Press, 1999.
- [Johnson, 1999] Johnson, C. K. Crystallographic topology 2: Overview and work in progress. In Alexiades, V. and Siopis, G., eds., *Trends in Mathematical Physics*, pp. 275–306. AMS/International Press, 1999.
- [Milnor, 1963] Milnor, J. *Morse Theory*, vol. 51 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, NJ, 1963.
- [Pasko *et al.*, 1995] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer* 11(8), 1995, pp. 429–446.
- [Stander & Hart, 1997] Stander, B. T. and Hart, J. C. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Computer Graphics (Annual Conference Series)*, Aug. 1997, pp. 279–286.
- [Witkin & Heckbert, 1994] Witkin, A. P. and Heckbert, P. S. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)*, July 1994, pp. 269–277.



SIGGRAPH2005

# Compactly Supported RBFs in the Management of Implicit Surfaces

Terry S. Yoo

Office of High Performance Computing and Communications

National Library of Medicine, NIH



SIGGRAPH2005

## Acknowledgements

- Volume Modeling Consortium
  - Bryan Morse (BYU)
  - K.R. Subramanian (UNCC)
  - Penny Rheingans (UMBC)
  - Kathleen Hoffman (UMBC)
  - David T. Chen (NLM/NIH)
  - Terry S. Yoo (NLM/NIH)
- Also
  - Greg Turk (GA Tech)
  - James F. O'Brien (UCB)

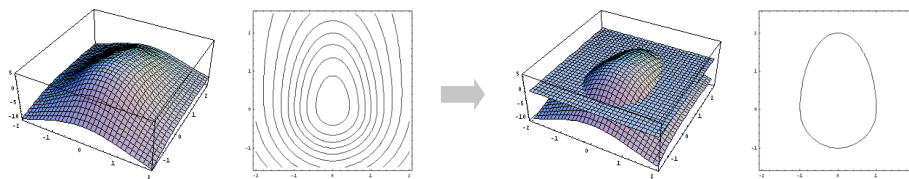


## Outline



- Summarize methods for constrained implicit surfaces
- Some successful results
- Examine the problem posed by large data
  - Deficiencies of thin-plate splines in medical modeling
- Propose alternative radial basis functions
- Results
- Future directions

## Implicit Surfaces



- Build a embedding or characteristic function.
- One connected isosurface is the implicit surface.
- Related to research in level sets.
- Not an parametric or polygonal surface representation.

## Related Work



- Hoppe 1992
- Witkin and Heckburt 1994
- Savchenko, *et al.* 1995
- Turk and O'Brien 1999
- Yngve and Turk 1999
- Morse, *et al.* 2001
- Carr, *et al.* 2001

## Implicit Surfaces that Interpolate



- Savchenko, *et al.*, 1995.
  - Interpolate surfaces from scattered points or contours.
  - Build their model from Green's function.
- Turk and O'Brien, 1999
  - "Variational Implicit Surfaces"
  - Build their model from thin plate splines.
  - Demonstrated their approach for morphing or shape interpolation.

## Variational Implicit Surfaces

Turk and O'Brien, 1999



SIGGRAPH2005

- Given: a set of oriented points (constraints)
  - surface scanners, segmented medical data, etc.
- Build a linear combination of radial basis functions (RBF) to create an embedding function.
- Choice of RBF: thin plate splines
  - Interpolates across relatively large distances.
  - Energy min. thin plate deformation. (variational)

## Thin-Plate Splines



SIGGRAPH2005

- Used by Turk and O'Brien (1998, 1999), Carr 2001
- Interpolating functions that minimize smoothness metric ("bending energy")

$$E(f) = \int_{s \in \Omega} f_{xx}^2(s) + 2f_{xy}^2(s) + f_{yy}^2(s) ds$$

- Solved using radial basis functions (Duchon, 1978)

## Variational Implicit Surfaces (continued)



Given:  $C = \{\bar{c}_1, \bar{c}_2, \dots, \bar{c}_n\}$

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

$$\phi(r) = r^3$$

3D

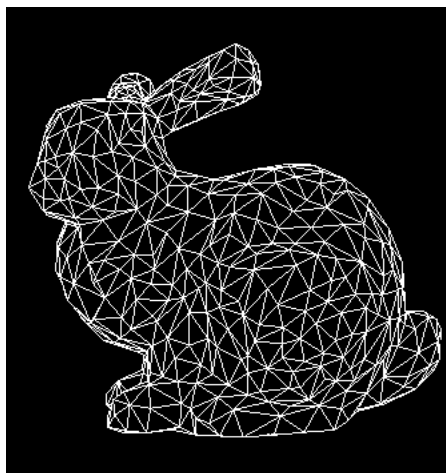
$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\phi(r) = r^2 \log r$$

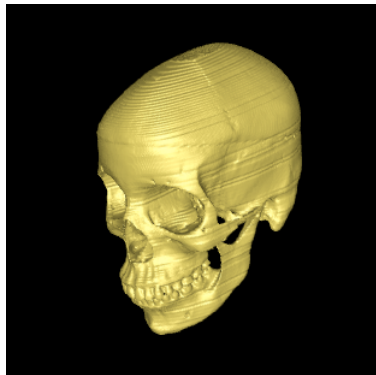
2D

$$\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

## Example: Coarse Model



## How Many Points?



- Subject: Dry Skull
- Slices: 124, 512x512
- Vertices:
  - 584,631
- Triangles:
  - 1,169,608

## Issues



- Dense data increases computational load.
- Thin plate spline interpolant limited to moderate numbers of (20,000) constraints.
- This approach not appropriate for complex models or large data.
  - Dense modalities (CT, MRI, etc.)
  - Complex surfaces: BRAIN!
- Carr, *et al.* 2001, limit the number of constraints

## Interpolating Functions



- What makes a good interpolation?
  - Must go through (or near) known values.  
For known constraints  $c_i$  with values  $h_i$ :

$$f(c_i) = h_i$$

- Should be “smooth”

## Radial Basis Functions



- Use circularly-symmetric functions  $\phi(r)$
- Interpolate using weighted sum:

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

- where
  - $\phi(r)$  is the radial basis function (many possible)
  - $\mathbf{c}_i$  are the centers of each component
  - $d_i$  are the weights of each component
  - $P(\mathbf{x})$  is a linear polynomial

## Linear System for RBFs



- Linear system of equations:

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

- Solve  $Ax = b$

Unknown

Unknown

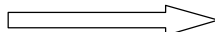
- A is symmetric, positive definite
- Guaranteed to have a solution
- A is  $(n + d+1)^2$  for n points in d dimensions
- May not be fast to solve for large n!

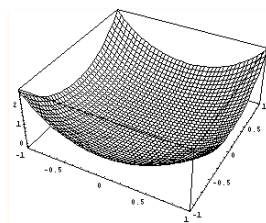
## Thin-plate Spline Implementation (revisited)



Given:  $C = \{\bar{c}_1, \bar{c}_2, \dots, \bar{c}_n\}$

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

In 3D  $\phi(r) = r^3$  



Solve  
for d's  
and p's

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Note:

$$\phi(0) = 0$$

$$\phi(r \gg 0) = r^3 \gg 0$$

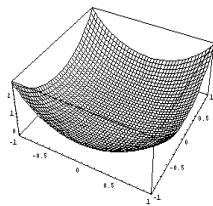
Matrix is FULL!

## Overview of Algorithm

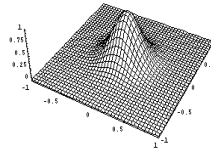


- Build matrix of  $\phi(c_i - c_j)$ 
  - $O(n^2)$  calculation
  - $O(n^2)$  storage
- Solve matrix
  - $O(n^3)$  calculation
  - $O(n^2)$  calculation possible
- Extract implicit surface
  - $O(n)$  per sample
- Skull Example
  - 584,632 vertices
  - 584,632 normals
  - 3 Dimensions
  - Matrix is:  
1,169,268 x 1,169,268 x  
IEEE floating point word  
= 5.468 Terabytes

## Approach: Change of RBF



Thin plate spline interpolant



Compactly supported interpolant

- Thin-plate spline interpolant has infinite support
  - not band limited
- Creates dense matrices, difficult to solve.
- Shift to compactly supported RBF with high continuity.



## Other Radial Basis Functions



- Other RBFs minimize other functionals.
- Gaussian:
  - Radially symmetric
  - Smooth result
  - Decays to zero rather than increasing
  - Effect of farther-away points is *less* rather than *more*
  - Matrix solution is better conditioned

## Compact Local RBFs



- Can go one step further and use *compact, locally-supported radial basis functions*.
- Can't just use arbitrary functions
  - Smooth
  - Differentiability of resulting interpolation
  - *Must produce positive definite matrix*

## Compact Local RBFs



- Wendland (1995) has solved for minimum-degree compact functions that guarantee that the solution matrix is positive definite.
- All of the form (can be scaled if needed)

$$\varphi(r) = \begin{cases} (1-r)^p P(r) & \text{if } |r| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

## Compact Local RBFs



- Major implication: radius of support  $\alpha$  imposes strict locality of solution.
- Advantages during all phases of using interpolating implicit surfaces
  - Building matrix
  - Solving matrix
  - Evaluating embedding function



SIGGRAPH2005

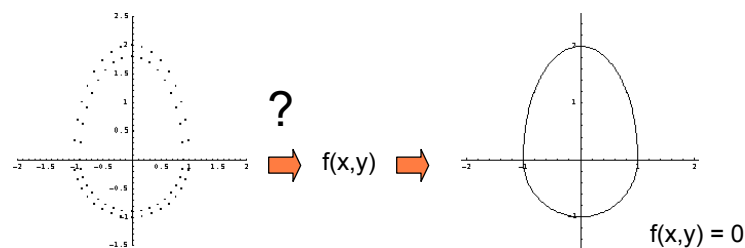
## Building the Matrix

- Don't need to compute value of RBF between all pairs of points, only those within  $\alpha$  of each other.
- Can use spatial data structure such as k-d tree to find “nearby” points in  $\log n$  time.
- Calculation is  $O(n \log n)$  vs.  $O(n^2)$
- Sparse storage:  $O(n)$  vs.  $O(n^2)$



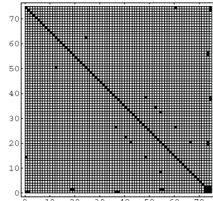
SIGGRAPH2005

## An Egg-sample

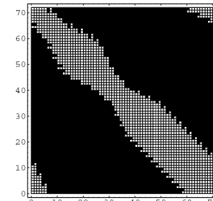


- Egg shape, 36 points, 36 offset normals

## Compactly Supported RBF



Thin plate spline interpolant matrix



Compactly supported interpolant matrix

- Local nature of the interpolants allows for spatial subdivision.
- Encourages the use of advanced data structures (K-D trees for spatial subdivision).
- Faster solvers for sparse matrices.

## Solving the Matrix



- Sparse matrix solution
  - Varies with the number of non-zero elements, not the absolute size of the matrix
  - Better conditioned
- Calculation is  $\sim O(n^{3/2})$  vs.  $O(n^3)$  or  $O(n^2)$

## Evaluating the Function



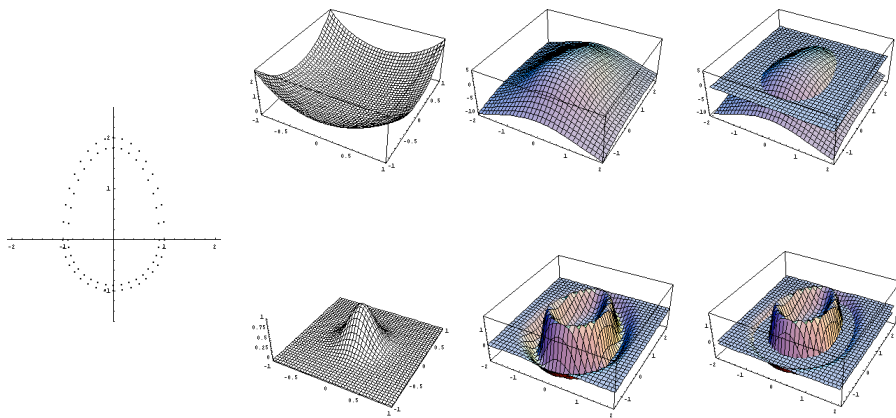
- Don't need to compute value of RBF for all constraints, only those within  $\alpha$  of the point being evaluated.
- Again use spatial data structure such as k-d tree to find "nearby" points in  $\log n$  time.
- Calculation is  $O(\log n)$  vs.  $O(n)$  to perform a single evaluation.

## Overview of Algorithm

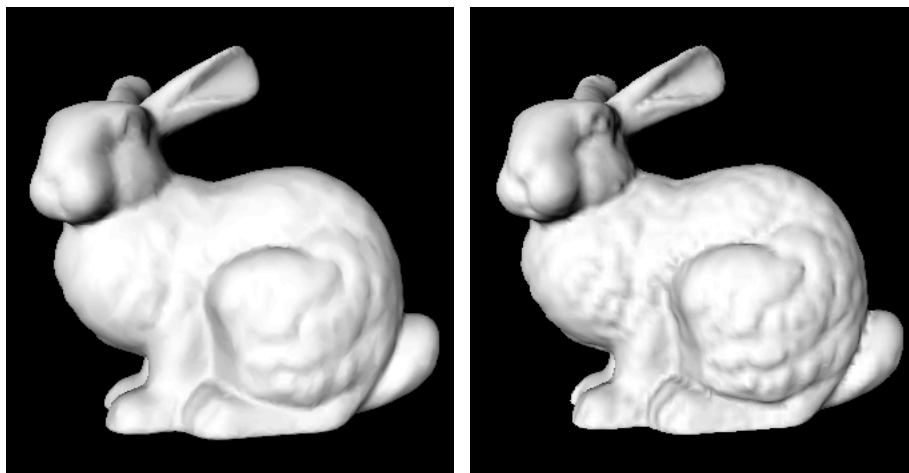


- Build matrix of  $\phi(c_i - c_j)$  only if  $c_i - c_j < \alpha$ 
  - $O(n \log n)$  calculation (using k-d tree)
  - $O(n k)$  storage  
where  $k$  = avg. number of points in support
- Solve matrix
  - $O(n k)$  calculation
- Extract implicit surface
  - $O(\log n)$  per function eval. (using k-d tree)

## Comparing TPS and CSRBFs



## 8000 Interpolated to 41,864



## Timing: Computing the Embedding Function

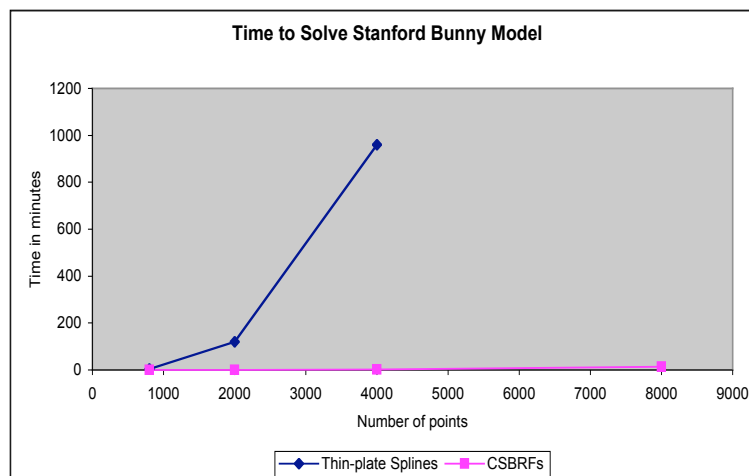


"Bunny" model	TPS	CSRBF
800 points	3:22 min	7.7 sec
2000	2 hrs	30.6 sec
4000	16 hrs*	132.8 sec
8000	n/a**	864 sec*

\* Significantly affected by swapping

\*\* 2GB storage required

## Timing: Computing the Embedding Function



## Why So Much Faster?



- Number of points in matrix

Points	TPS	CSRBF	%
800	646,416	69,582	10.8%
2000	4,016,016	220,095	5.5%
4000	16,032,016	850,675	5.3%
8000	64,064,016	3,342,253	5.2%

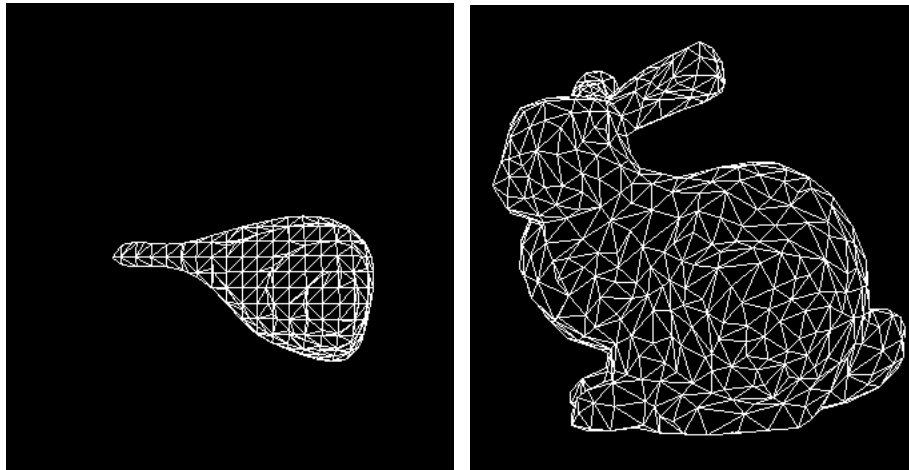
## Something to careful of...



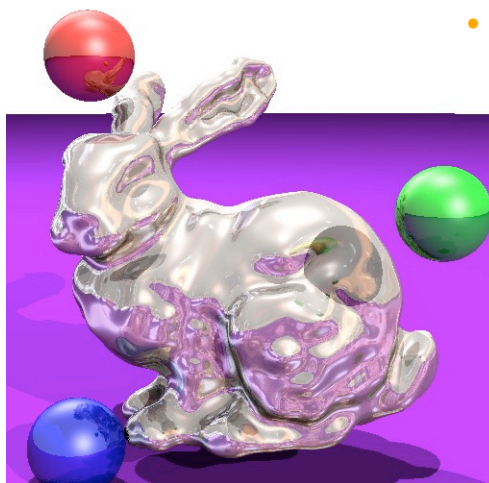
- Compact, locally-supported RBFs produce
  - Desired zero set for implicit surface
  - Lots of other zeroes inside/outside one radius of support from the surface!
- Have to be careful about seeding, extracting the implicit surface



## Example: Inner Hull



## Outer hull... interfering with ray tracing?



- Ray tracing is possible

- Adjust the level set to suppress the outside isosurface

OR

- Apply a transfer function to reflect only at zero crossings with high gradient magnitude

## Other Implications for Future Work



- Faster method means interpolating implicit surfaces are viable for medical models
  - Resampling models
  - More accurate model simplification
- Local support means local effect
  - Incremental updating
  - Interactive modeling

## Future Work



- Error analysis vs. thin-plate splines
- More efficient data structures for managing spatial locality
- Other types of sparse solvers
  - Using LU, could use SVD, CG, etc.
- Better ways to deal with extracting desired isosurface and not the inner/outer hulls
- More timing studies/profiling

## Conclusions

---



- Choice of the RBF depends on application (number of points).
- A little computer science goes a long way.
- Leads to interesting interdisciplinary work.
  - computer vision, computational geometry
  - computer graphics, visualization, medicine

# Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions

Bryan S. Morse<sup>1</sup>, Terry S. Yoo<sup>2</sup>, Penny Rheingans<sup>3</sup>, David T. Chen<sup>2</sup>, K. R. Subramanian<sup>4</sup>

<sup>1</sup>Department of Computer Science, Brigham Young University  
morse@byu.edu

<sup>2</sup>Office of High Performance Computing and Communications, National Library of Medicine  
{yoo | dave}@nlm.nih.gov

<sup>3</sup>Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County  
rheingan@cs.umbc.edu

<sup>4</sup>Department of Computer Science, University of North Carolina at Charlotte  
krs@cs.uncc.edu

## Abstract

*We describe algebraic methods for creating implicit surfaces using linear combinations of radial basis interpolants to form complex models from scattered surface points. Shapes with arbitrary topology are easily represented without the usual interpolation or aliasing errors arising from discrete sampling. These methods were first applied to implicit surfaces by Savchenko, et al. and later developed independently by Turk and O'Brien as a means of performing shape interpolation. Earlier approaches were limited as a modeling mechanism because of the order of the computational complexity involved. We explore and extend these implicit interpolating methods to make them suitable for systems of large numbers of scattered surface points by using compactly supported radial basis interpolants. The use of compactly supported elements generates a sparse solution space, reducing the computational complexity and making the technique practical for large models. The local nature of compactly supported radial basis functions permits the use of computational techniques and data structures such as *k*-d trees for spatial subdivision, promoting fast solvers and methods to divide and conquer many of the subproblems associated with these methods. Moreover, the representation of complex models permits the exploration of diverse surface geometry. This reduction in computational complexity enables the application of these methods to the study of shape properties of large complex shapes.*

## 1 Introduction

As a research field of growing interest, implicit surface representations have a colorful history, with their founda-

tions established early in the development of 3D curves and surfaces in computer graphics [2]. However, the computation of implicit surfaces has often been hampered by the constraints of available processing power and the limited complexity of the models that can be created. As CPU speeds, available memory, and computing costs have evolved, more complex models and techniques have become possible, spurring general interest in implicit surfaces [3, 6]. However, managing complex models remains difficult. The core research in modeling with implicit surfaces centers around primitives that do not always facilitate the control of the exact surface. While alternatives to surface control through simple primitives exist, they also have drawbacks. In their recent work on adaptive particle sampling and control of implicit surfaces, Heckbert and Witkin [18] report that particle control of such surfaces is slippery and elusive.

Recent growth in level set techniques [9, 12] and variational methods [7] have created new interest in understanding and manipulating complex surface models directly from the surface representation. Whitaker and Breen [17] have shown how level set techniques can be used to model and manipulate computer graphic shapes, effectively morphing from one to another. They characterize the level set as an implicit representation where the primitives are distributed throughout an active volumetric cloud layer near the surface boundary. They utilize Sethian's notion of the active set, to reduce the size of the volume representation to a sparse collar of primitives. However, even with this reduction, the representations are cumbersome, and the bookkeeping to support these methods are intricate and difficult to maintain.

Other recent work has developed techniques for interpolating an implicit surface directly from surface point data [11, 14]. This work provides some insight into how to

manage and employ a collection of implicit primitives while simultaneously directly controlling the surface parameters. This method allows direct specification of a complex surface from sparse, irregular surface samples. The method is quite flexible and has been extended to higher dimensions to support shape interpolation [15]. However, because of computational and storage complexity, the technique as described cannot be used to model surfaces where large numbers of surface points are included, making it unsuitable for applications where range data or tomographic reconstruction often lead to data described by hundreds of thousands of surface points.

This paper primarily addresses the topic of computational complexity. We explore an adaptation of the methods in [11, 14], applying compactly supported radial basis functions [16] to create an efficient algorithm for computing interpolated surfaces. Our technique produces significant improvements in memory utilization and computational efficiency. We discuss both the advantages of our technique as well as the consequences or prerequisite requirements imposed by our methods in later sections.

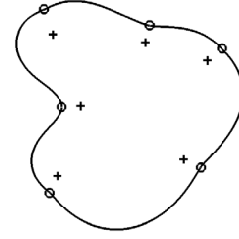
This more efficient approach is creating opportunities to explore complex shapes through implicit modeling methods. In particular, the thin-plate-spline radial basis function and the Green’s function solutions with their order  $O(n^2)$  solutions are impractical when the number of constraints exceeds a few thousand points. By shifting to a compactly supported radial basis function, we can create differentiable analytic representations of large complex models. These efficient solutions make possible techniques for studying the deep structure of solid shapes. In the discussion section of this paper, we present early applications of these implicit surfaces to problems such as surface shape analysis.

## 2 Background / Problem

The key idea in both [11] and [14] is that one may produce an implicit surface from known surface points by interpolating the embedding function within which the surface is implicitly defined. While we primarily follow here the presentation found in [14], we also encourage the interested reader to see an alternative and earlier formulation in [11].

### 2.1 Interpolating Surfaces by Interpolating Embedding Functions

An *implicit surface* is defined by  $\{\bar{x} : f(\bar{x}) = 0\}$  for some embedding function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$ . The key idea behind interpolated implicit surfaces is to find a smooth embedding function  $f$  such that  $f(\bar{x}_i) = 0$  for each known surface point  $\bar{x}_i$ , and  $f(\bar{y}_i) = 1$  for one or more points  $\bar{y}_i$  known to be inside the shape. (Alternatively, constraints of the form  $f(\bar{y}_i) = -1$  may be added for points outside the



**Figure 1. Implicit curve interpolated using zero-constraints along the curve and positive constraints just inside these points in the direction opposite the known (or desired) normals. (Figure courtesy of Greg Turk.)**

shape.) Turk and O’Brien select these interior points using normals at the surface points as shown in Figure 1.

This may be generalized to a *scattered-data interpolation* problem as follows. Given a set of positions  $\bar{c}_i$  and corresponding values  $h_i$ , solve for an embedding function  $f$  such that  $f(\bar{c}_i) = h_i$ . Thus, surface interpolation may be turned into higher-dimensional scattered-data interpolation, a well-studied field.

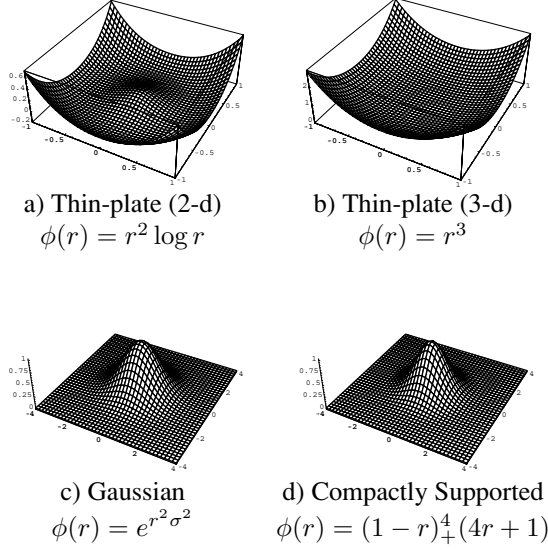
Turk and O’Brien chose to use *thin-plate splines*, which minimize the bending energy of the embedding function:  $E = \int_{\Omega} f_{xx}^2(\bar{x}) + f_{xy}^2(\bar{x}) + f_{yy}^2(\bar{x}) d\bar{x}$ . They called their method *variational implicit surfaces*, because they formulate the problem as one of variational interpolation (minimizing an energy functional subject to interpolative constraints). They did not, however, solve for the embedding function using an iterative minimization approach but instead solve for the known closed-form solution using radial basis functions as described in Section 2.2. (Savchenko, et al. use similar splines based on the use of Green’s function.)

### 2.2 Radial Basis Functions

Scattered data interpolation can be achieved using *radial basis functions* centered at the constraints. Radial basis functions are circularly-symmetric functions centered at a particular point.

Duchon [5] has shown that solving for thin-plate splines through known points in two dimensions is equivalent to interpolating these points using the biharmonic radial basis function  $\phi(r) = r^2 \log |r|$  (Figure 2a). In three dimensions, the thin-plate solution is equivalent to interpolating these points using the radial basis function  $\phi(r) = |r|^3$  (Figure 2b).

Radial basis functions may be used to interpolate a function with  $n$  points by using  $n$  radial basis functions centered at these points. The resulting interpolated function thus be-



**Figure 2. Comparison of different radial basis functions**

comes

$$f(\bar{x}) = \sum_{i=1}^n d_i \phi(\|\bar{x} - \bar{c}_i\|) \quad (1)$$

where  $\bar{c}_i$  is the position of the known values,  $d_i$  is the weight of the radial basis function positioned at that point. In some cases (including the thin-plate spline solution), it is necessary to add a first-degree polynomial  $P$  to account for the linear and constant portions of  $f$  and ensure positive-definiteness of the solution:

$$f(\bar{x}) = \sum_{i=1}^n d_i \phi(\|\bar{x} - \bar{c}_i\|) + P(\bar{x}) \quad (2)$$

To solve for the set of weights  $d_i$  that satisfy the known constraints  $f(\bar{c}_i) = h_i$ , we substitute each  $\bar{c}_i$  into Eq. 2:

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) = h_i \quad (3)$$

or, if a polynomial is required:

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i \quad (4)$$

Solving for the weights  $d_j$  using Eq. 3 and denoting  $\phi_{ij} = \phi(\|\bar{c}_i - \bar{c}_j\|)$  produces the following system:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} \quad (5)$$

If a polynomial is required, Eq. 4 similarly becomes

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (6)$$

In both Eqs. 5 and 6 the matrix is obviously real symmetric, and with proper selection of basis functions it can be made positive-definite. Thus, a solution always exists to these systems.

### 2.3 Algorithmic Complexity

Calculating and using implicit surfaces that interpolate may be analyzed in three parts:

1. Constructing the system of equations,
2. Solving the system of equations, and
3. Evaluating the interpolating function (as required).

#### 2.3.1 Constructing the System of Equations

A significant portion of the computational cost involved in calculating these implicit surfaces is the cost required to construct the matrix (or submatrix)  $\phi_{ij} = \phi(\|\bar{c}_i - \bar{c}_j\|)$ . Recall that the thin-plate radial basis function is  $\phi(r) = r^2 \log r$  (two dimensions) or  $\phi(r) = r^3$  (three dimensions). This means that *the matrix is entirely non-zero except along the diagonal*, requiring the calculation of all inter-point distances within the set  $\{\bar{c}_i\}$ . Although the symmetry of the matrix cuts the computational cost in half, the computational complexity is still  $O(n^2)$ . Furthermore, storage of such a matrix requires  $O(n^2)$  floating-point values—a potentially more prohibitive factor than the computational complexity.

#### 2.3.2 Solving the System of Equations

Although Turk and O'Brien use LU factorization (an  $O(n^3)$  algorithm) to solve Eq. 5, they correctly point out that it is possible to solve this system in  $O(n^2)$  by iterative means. Thus, while solution of the system may appear to be the limiting step, it need only be as computationally expensive as constructing the system.

#### 2.3.3 Evaluating the Function

For nearly all applications it is not enough to simply solve for the weights of the respective radial basis functions.

Rather, it is necessary to evaluate this embedding function at potentially many points in order to extract the isosurface, calculate normals or other derivative quantities, etc. Because the terms  $\phi(\|\bar{x} - \bar{c}_i\|)$  in Eq. 1 are all non-zero for the thin-plate solution (except for one zero term when  $\bar{x} \in \{\bar{c}_i\}$ ), all of the terms must be used in calculating any one point. Thus, each evaluation of the interpolated function is  $O(n)$ .

## 2.4 Problems with the Thin-Plate Solution

While the thin-plate spline embedding function does indeed minimize bending energy, it has the following drawbacks in computation and usefulness for user interaction:

1.  $O(n^2)$  computation is required to build the system of equations.
2.  $O(n^2)$  storage is required (for the nearly-full matrix) to represent the system.
3.  $O(n^2)$  computation is required to solve the system of equations.
4.  $O(n)$  computation is required per evaluation
5. Because every known point affects the result, a small change in even one constraint is felt throughout the entire resulting interpolated surface, an undesirable property for shape modelling.

## 3 Using Compactly-Supported Radial Basis Functions

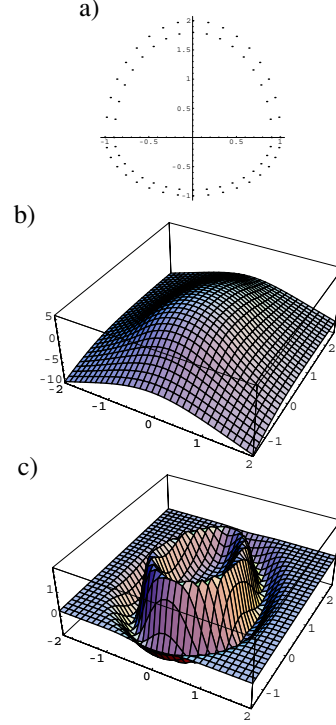
Wendland [16] has recently solved for the minimum-degree polynomial solution for compact, locally-supported radial basis functions that guarantee positive-definiteness of the matrix (Figure 2d). All of the solutions have the form

$$\phi(r) = \begin{cases} (1-r)^p P(r) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

For various degrees of desired continuity ( $C^k$ ) and dimensionality ( $d$ ) of the interpolated function, he has derived the following radial basis functions:

$d = 1$	$(1-r)_+$	$C^0$
	$(1-r)_+^3(3r+1)$	$C^2$
	$(1-r)_+^5(8r^2+5r+1)$	$C^4$
$d = 3$	$(1-r)_+^2$	$C^0$
	$(1-r)_+^4(4r+1)$	$C^2$
	$(1-r)_+^6(35r^2+18r+3)$	$C^6$
	$(1-r)_+^8(32r^3+25r^2+8r+1)$	$C^6$
$d = 5$	$(1-r)_+^3$	$C^0$
	$(1-r)_+^5(5r+1)$	$C^2$
	$(1-r)_+^7(16r^2+7r+1)$	$C^4$

These functions have radius of support equal to 1. Scaling of the basis functions (i.e.,  $\phi(r/\alpha)$ ) allows any desired radius of support  $\alpha$ .



**Figure 3. A simple 36-point ovoid (a) interpolated using thin-plate (b) and compactly-supported (c) radial basis functions.**

### 3.1 Example

Figure 3 illustrates using both thin-plate and compactly-supported radial basis functions to compute embedding functions. The constraint points consist of 36 points in an ovoid shape with 36 normal (positive valued) constraints placed just inside (3a). A thin-plate radial basis function produces a globally-smooth embedding function (3b). A compactly-supported radial basis function produces an embedding function that does not have global smoothness but is as smooth as the thin-plate spline interpolation in a narrow band surrounding the shape both inside and out.

### 3.2 Algorithm

Using compactly-supported radial basis functions provides advantages in all three phases of the implicit-surface interpolation algorithm.

#### 3.2.1 Constructing the system

Because the radial basis functions have finite support,  $\phi(\|\bar{c}_i - \bar{c}_j\|) = 0$  for all  $(\bar{c}_i, \bar{c}_j)$  farther apart than the radius of support. By using a  $k$ -d tree [1], the set of all points

within distance  $r$  of a particular point  $\bar{c}_i$  can be determined in  $O(\log n)$  time.

A  $k$ -d tree is a multidimensional binary tree with the following sorting property for a tree with point  $\bar{x}$  at the root and subtrees  $T_{\text{left}}$  and  $T_{\text{right}}$ .

$$\begin{aligned}\forall \bar{y} \in T_{\text{left}} : \bar{y}^d &\leq \bar{x}^d \\ \forall \bar{y} \in T_{\text{right}} : \bar{y}^d &> \bar{x}^d\end{aligned}$$

where the sorting dimension  $d$  changes at each level of the tree.

$k$ -d trees can be used to find all points within distance  $r$  of a particular constraint in  $O(n \log n)$  time using the following algorithm (C pseudocode):

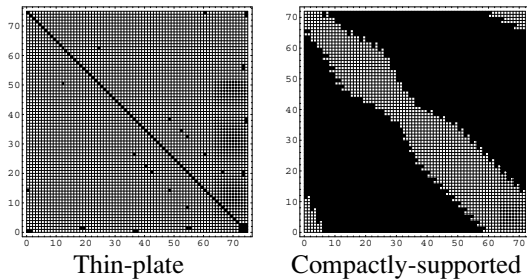
```
void Search (KDtree T, Point P, int radius)
{
    if (T->point[T->dim] < P[dim] + radius)
        Search(T->right);
    if (T->point[T->dim] > P[dim] - radius)
        Search(T->left);
    Test(T->point, P, radius);
}
```

While a number of points must still be tested explicitly, the multidimensional sorting nature of the  $k$ -d tree allows a large number of points to be rejected at each level of the tree.

The resulting matrix is extremely sparse, as shown in Figure 4. Using a sparse-matrix representation (we use the Hartwell-Boeing format), only  $O(n)$  storage is required.

### 3.2.2 Solving the system

If the average number of points within the radius of support of each constraint  $\bar{c}_i$  is less than some constant  $k$ , the number of non-zero entries in the matrix is  $O(n)$ . We use a direct (LU) sparse matrix solver [4] to find the solution to



**Figure 4. Structure of the matrices produced by thin-plate and compactly-supported radial basis functions (Figure 3). The compactly-supported basis function produce a matrix that is sparse (black), while the thin-plate basis functions produce a matrix that is nearly full (white).**

the system of equations. The computational complexity of such a solver depends on the amount of matrix “fill in” that occurs during the solution, but some authors have reported behavior in the range  $O(n^{1.2})$  to  $O(n^{1.5})$  [10]. Our own experience (Section 4) agrees with this.

### 3.2.3 Evaluating the interpolating function

We can exploit the spatial locality of the compactly-supported radial basis functions during evaluation of the embedding function  $f$  by recognizing that only a fraction of the terms of Eq. 1 are non-zero for a given  $\bar{x}$ :  $\phi(\bar{c}_i - \bar{c}_j) \neq 0$  iff  $\|\bar{c}_i - \bar{c}_j\| < 1$ . By again using a  $k$ -d tree to organize the constraints spatially, each evaluation of the interpolating function requires only  $O(\log n)$  operations to determine these non-zero terms.

## 3.3 Thin-Plate vs. Compactly-Supported Radial Basis Functions

Using compactly-supported radial basis functions directly addresses each of the five problems identified previously with the thin-plate spline basis functions (Section 2.4) as follows:

	Thin-plate	Compact
Computation to build	$O(n^2)$	$O(n \log n)$
Computation to solve	$O(n^2)$	$O(n^{1.5})$
Storage to build/solve	$O(n^2)$	$O(n)$
Computation to evaluate	$O(n)$	$O(\log n)$
Effect of a single point	Global	Local

## 4 Results

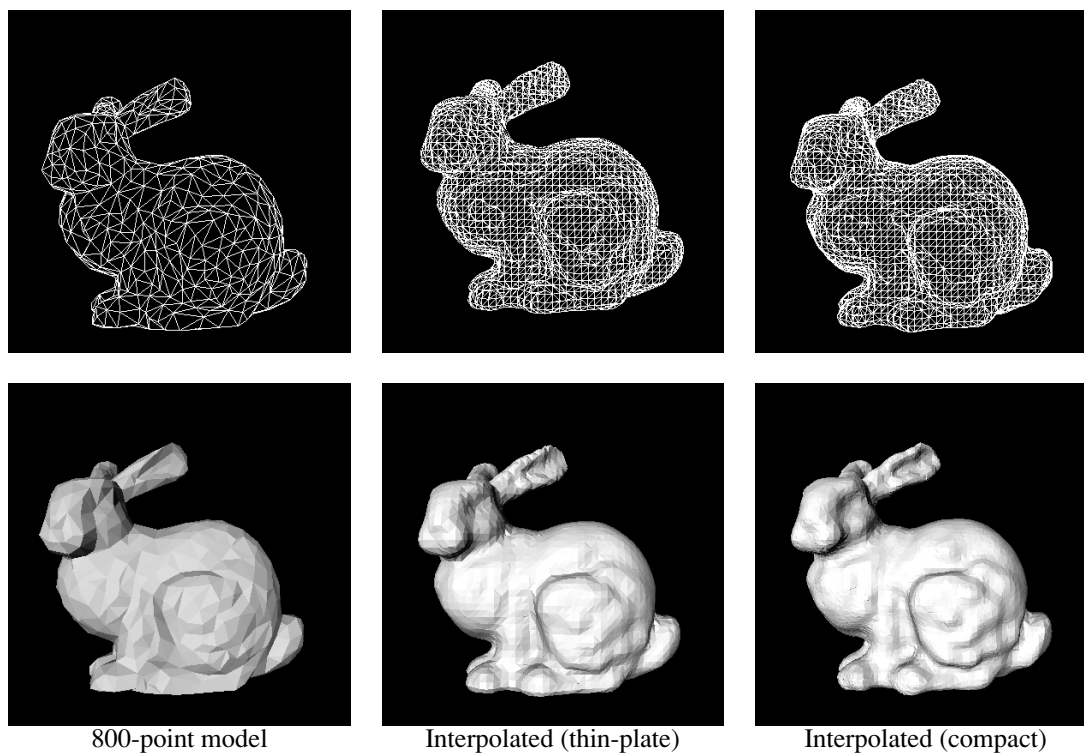
Figure 5 compares the results of interpolating an 800-point model of the Stanford bunny using both thin-plate and compactly-supported radial basis functions. The results of the two methods are qualitatively identical.

Interpolation of larger models is possible using compactly-supported radial basis functions. The 8000-point model in Figure 6 was interpolated using compactly-supported radial basis functions, and the resulting isosurface was tessellated to 41,864 points.<sup>1</sup> The interpolated image shows sharpness and detail hidden by the flat polyhedra of the original model.

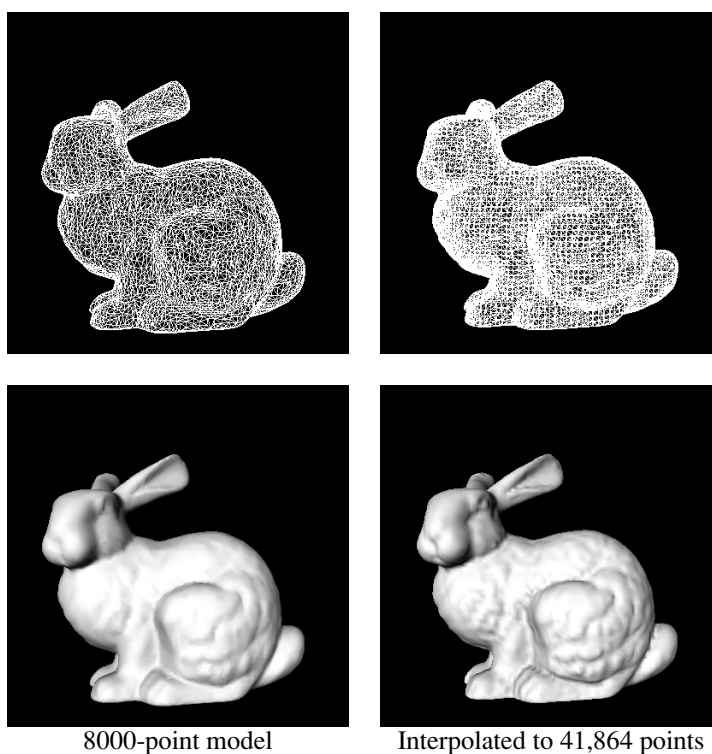
Table 1 summarizes the time required to calculate various differing-resolution models of the same figure (Stanford bunny). The radius of support used for each model was selected so as to keep the number of points in the radius of

<sup>1</sup>A similar thin-plate interpolation would require almost 2 GB of storage for the matrix alone and could not be calculated on our system.





**Figure 5. Comparison of original model to implicit surfaces extracted from embedding functions calculated using both thin-plate and compactly-supported basis functions. The results of the interpolations are qualitatively identical.**

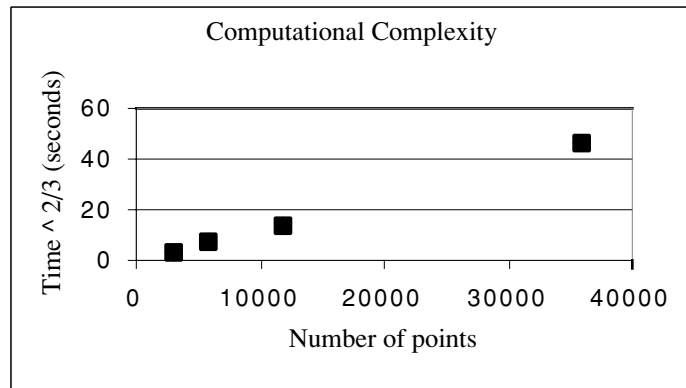


**Figure 6. Interpolation of an 8000-point model of the Stanford bunny**

Points	Constraints	Radius	Non-zero	Per Row	% Full	Build $k$ -d	Build Matrix	Solve Matrix	Total
2922	5848	0.200	476173	81.4	1.39%	0.02	1.05	5.93	7.00
5839	11682	0.150	973423	83.3	0.71%	0.04	2.09	17.29	19.42
11831	23666	0.100	1663733	70.3	0.30%	0.08	4.00	45.44	49.52
35947	71898	0.004	5061542	70.4	0.10%	0.93	31.51*	284.01*	316.45*

\* some virtual memory swapping

**Table 1. Execution times in seconds for various phases of computation for interpolated implicit surfaces using compactly-supported radial basis functions.**



**Figure 7. Timing results for four different-resolution models of the same figure (Table 1). The linear graph when plotted on a  $time^{\frac{2}{3}}$  vertical axis indicates  $O(n^{1.5})$  complexity.**

Thin-Plate			Compactly-Supported		
Points	Build	Solve	Points	Build	Solve
800	0.58 sec	2.56 min	800	0.57 sec	1.53 sec
2000	6.3 sec	2:05 hrs	2922	1.07 sec	5.93 sec
4000	25.0 sec	16:11 hrs*	5839	2.13 sec	17.29 sec
8000	n/a (1)	n/a	11831	4.08 sec	97.53 sec
35947	n/a (2)	n/a	35947	31.44 sec	284.01 sec

\* significant virtual memory swapping

**Table 2. Comparison of execution times required to calculate embedding functions using thin-plate vs. compactly-supported radial basis functions**

Thin-Plate		Compactly-Supported	
Points	Memory (MB)	Points	Memory (MB)
800	19.5	800	1.0
2000	122.1	2922	3.6
4000	488.3	5839	7.4
8000	1,953.1	11831	12.7
35947	39,434.4	35947	38.6

**Table 3. Comparison of memory requirements (matrix only, double-precision floating-point values) required to calculate embedding functions using thin-plate vs. compactly-supported radial basis functions**

support (approximately) constant, thus allowing comparison of the results. The dominant term in the required computation is the time to solve the system of equations, which seems to demonstrate  $O(n^{1.5})$  complexity (Figure 7).

Table 2 compares these running times to the time required to calculate comparable thin-plate interpolations (using same-size or smaller models). The  $O(n^2)$  time required to compute these models using thin-plate radial basis functions quickly becomes prohibitive.

Similarly, Table 3 compares the memory required to represent the matrix for the models described in Table 2. As with the computational complexity, the  $O(n^2)$  storage required to compute these models using thin-plate radial basis functions also quickly becomes prohibitive.

## 5 Considerations

The finite nature of the compactly-supported radial basis functions introduces two factors that must be considered when using them to interpolate embedding functions.

### 5.1 Selecting the Radius of Support

The finite radius of support introduces an additional parameter that doesn't exist in the thin-plate implementation. Proper selection of the radius of support is critical to achieving optimal efficiency of computation and results. Too small a radius can produce basis functions that are unable to span the inter-constraint gaps. Too large a radius does not adversely affect the results but reduces the sparseness of the matrix, thus increasing the computation required. It is thus necessary to select a radius of support that is both large enough to produce effective results and not so large that the computation becomes impractical.

### 5.2 Isosurface Extraction

Because of the finite extent of the compactly-supported radial basis functions, only those points within the radius of support of one of the original positions have non-zero values. For all points outside this band, all of the terms of Eq. 1 are zero. (Figure 3c illustrates this.) In this way, these embedding functions are not the same as those normally used for implicit surfaces—the implicit surface represented is not the only set of zero-valued points in the space. However, the implicit surface does form a unique contiguous locus of zero-valued points passing through the constraints. In this sense, the method presented here is somewhat similar to the narrow-band active set approach of Sethian [12] or Whitaker and Breen [17].

An isosurface extractor may be used to extract this surface by seeding it with any one of the initial constraints. However, care must be taken so that the step size of the

extractor does not cause it to jump outside the band of non-zero points. It is, however, rather easy to explicitly recognize when no non-zero terms are found in Eq. 1 (none of the constraint points lie within the radius of support of the point being evaluated).

Because the surface of interest is not the only zero set of the embedding function, the resulting embedding function has limited application in CSG, interpolation [15], or similar applications. However, we are experimenting with a hybrid approach that interpolates a subset of the points using radial basis functions with infinite support and the difference using basis functions with compact support.

## 6 Discussion: Analytic Approaches Enabled by Efficient Algorithms

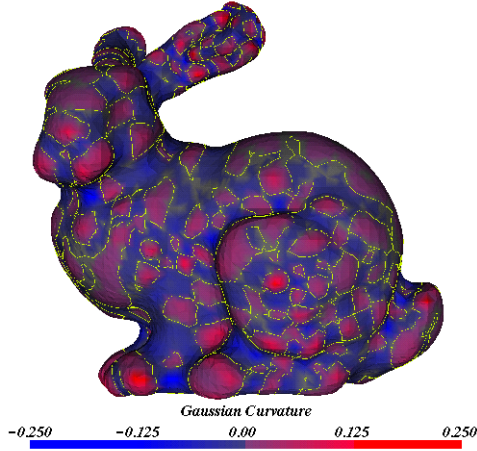
An efficient framework for finding embedding functions that define implicit surfaces from scattered data points increases the practicality of studying complex shape models represented by large numbers of such points. Models captured from physical phenomena usually contain large numbers of surface points, polygons, or other surface primitives that are not easily reduced. For instance, polygonal representations of medical data often begin with models containing millions of triangles, which can later be simplified to hundreds of thousands of polygons.

While the polygonal representations of these models can be rendered using current graphics hardware, the discrete sampling introduced by the process of producing the polygons raises barriers to deeper studies of the geometry of the surfaces themselves. Implicit methods solve this difficulty by creating analytic functions that smoothly reconstruct a surface from a constellation of points. In addition, the implicit surface constructed using the method presented here is differentiable. Local surface geometry now becomes approachable since numerically stable solutions can be found to sample higher order derivatives of the implicit surface.

### 6.1 Differential Geometry of Shape

Differential geometry is the study of multilocal surface behavior, employing the normal vector and tangent plane at each surface point as a reference environment. Curvature and other geometric features relative to the surface tangent are measured using second order derivatives [13]. Volumetric approaches can compute approximations to these measurements based on discretely sampled grids [12]. However, the accuracy and behavior of the sampled derivatives is subject to aliasing and sampling issues, exacerbated by the noise-amplifying effects of higher-order functions.

Implicit surfaces allow us to reconstruct smooth surface representations from a set of oriented points and sample



**Figure 8. Gaussian curvature computed over an analytically-defined implicit surface calculated from scattered surface points**

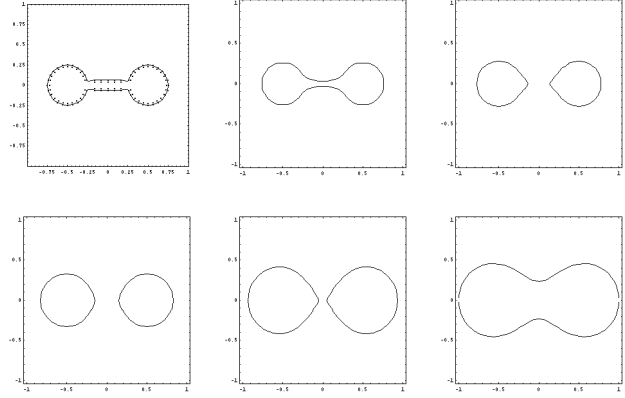
higher order derivatives with instantaneous precision. Analytic representations of not only surface shape, but also of the embedding function, simplifies the computation of the normal vector, the tangent plane, and the principal curvatures of the implicit surface at any arbitrary location.

Figure 8 shows the Gaussian curvature (the product of the principal curvatures) of an array of implicit surfaces calculated using the method presented here. Blue areas represent positive Gaussian curvature (elliptical regions) and red areas represent negative Gaussian curvature (hyperbolic or saddle-shaped regions). The yellow contour lines indicate the places where the Gaussian curvature is zero, separating the red and blue surface regions. The yellow contours denote parabolic curves where specular and diffuse highlights fuse or reproduce under changing lighting conditions.

Note: the polygonalization of the surface and the interpolation errors are artifacts of the visualization technique. The implicit surface itself can be sampled with arbitrary precision to generate smooth models with correspondingly smooth representations of curvature.

## 6.2 Future Directions: Scale Space, Ray Tracing, and Other Topics

Linear scale space filtering was introduced by Witkin [19] as a means of measuring the saliency of features within an image. Further work established the general field of scale-space theory in computer vision [8]. The fundamental notion of such analysis is that significant details of an image (or surface representation) persist as the scale or the measurement aperture is increased. In the pursuit of multiscale image descriptions, a Gaussian kernel is usually used as the measurement aperture function.



**Figure 9. A dumbbell figure reconstructed from sample points and represented at successively larger scales by convolving the embedding function with successively larger Gaussian kernels**

From this perspective, a scale space representation of the implicit models presented in this paper can be constructed as a convolution of a Gaussian kernel with the embedding function,  $f(\vec{x}) \otimes g(\vec{x}, \sigma)$ . Since convolution is both commutative and distributive with respect to addition, this is equivalent to convolving each of the radial basis interpolants with a Gaussian. This process can be approximated by solving for the implicit surface with a particular radius of support, and later dilating the compactly supported radial basis function during the evaluation to create a scale space level set representation of the basic function.

Figure 9 shows a 2-D implicit figure represented at a range of scales. The original model has been reconstructed from oriented points as an implicit surface. In the subsequent representations, the embedding function has been convolved with an approximation to a Gaussian kernel, and the implicit surface reconstructed. Fine details such as the corners and discontinuities associated with the cross member in the figure are suppressed at moderate scales. Eventually at the largest scales, the figure is viewed as a single topologically simple object. This approach to representing object shape may have applications in modeling and computer graphics in the representation of objects at multiple levels of detail.

Beyond the application of scale-space image-analysis techniques to implicit surface representations, there remain interesting problems of rendering these models. Implicit surfaces based on signed distance functions and other embedding functions with similar properties are easily rendered through ray tracing. Because of the multiple zero level sets created by the compactly supported radial basis function approach, a basic ray tracing method is insufficient for rendering these models. However, in the visual-

ization of discrete volume data, complex transfer functions that include geometric information such as gradient magnitude and isosurface curvature are able to capture surfaces based on features other than simple isovalues. Future work will include the development of transfer functions for ray-cast rendering of these models.

## 7 Conclusion

Given a set of points  $C = \{\bar{c}_1, \bar{c}_2, \dots, \bar{c}_n\}$  and associated normals, the method presented here interpolates an implicit surface through those points by finding a scalar embedding function  $f(\bar{x})$  whose zero level set,  $\{\bar{x} : f(\bar{x}) = 0\}$ , passes through all points in  $C$ . Following Turk and O'Brien [14] (see also [11]), we employ radial basis interpolating functions in a linear system of  $2n$  equations and  $2n$  unknowns, expressed as an  $2n \times 2n$  matrix. Unlike the original use of  $\phi(r) = |r|^3$  as the underlying interpolant, we use a family of radial basis functions with the necessary continuity but also with compact local support. The result is a sparse system whose solution can be accelerated. The result is a single, accelerated, closed form analytic representation of the desired surface.

The shift from a radial basis function of infinite extent to a compactly supported one creates dramatic gains in memory utilization and computational complexity. Previous work described solutions for systems of equations of order  $O(n^2)$  complexity. The shift to finite interpolants and sparse matrices has shifted the complexity of the matrix solution to order  $O(n^{1.5})$ , the loading of the matrix data structures to  $O(n \log n)$ , and the memory requirements to order  $O(n)$ . Evaluation of the interpolated embedding function is similarly reduced to  $O(\log n)$ .

These improvements in efficiency make possible a variety of applications that were previously impractical with an infinite radial basis function. We have briefly surveyed our first probes into the differential geometry of surface shape and explorations in scale space analysis of complex models using implicit surfaces interpolated from scattered surface data points.

## Acknowledgments

This work was performed in large part at the National Library of Medicine under a visiting faculty program supporting both Dr. Morse and Dr. Subramanian. Dr. Rheingans was supported in part by NSF CAREER Grant #9996043. We would like to thank Greg Turk for his useful conversations and for making his code available to us, upon which our implementation is based. We would also like to thank Dr. Michael Ackerman and the staff of NLM's Office of High Performance Computing and Communications for their help and support. Finally, we would like to thank the reviewers and program committee for SMI2001 for their many helpful suggestions.

## References

- [1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.
- [2] J. Blinn. A generalization of algebraic surface drawing. *IEEE Transactions on Graphics*, 1(3):235–246, 1982.
- [3] J. Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan-Kaufman, 1997.
- [4] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, Mar. 1990.
- [5] J. Duchon. Sur l'erreur d'interpolation des fonctions de plusieurs variables par les  $d^m$  splines. *R.A.I.R.O Analyse numerique*, 12(4):325–334, 1978.
- [6] J. Hart and e. D. Ebert. *New Frontiers in Modeling and Texturing*. Siggraph 97 Course Notes, 1997.
- [7] B. Kimia, A. Tannenbaum, and S. Zucker. On optimal control methods in computer vision and image processing. In B. t.H. Romeny, editor, *Geometry Driven Diffusion in Computer Vision*, pages 307–338. Kluwer, 1994.
- [8] T. Lindeberg. *Scale-space theory in computer vision*. Kluwer Academic Publishers, 1994.
- [9] S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation. *J. Comput. Phys.*, 79:12–49, 1988.
- [10] R. A. Saleh, K. A. Gallivan, M. Chang, I. N. Hajj, D. Smart, and T. N. Patrick. Parallel circuit simulation on supercomputers. *Proceedings of the IEEE*, 77(12):1915–1930, 1989.
- [11] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.
- [12] J. A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.
- [13] J. Thirion. New feature points based on geometric invariants for 3d image registration. Technical Report INRIA-RR-1901, INRIA, 1993.
- [14] G. Turk and J. F. O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1998.
- [15] G. Turk and J. F. O'Brien. Shape transformation using variational implicit surfaces. In *Computer Graphics Proceedings, Annual Conference Series*, 1999.
- [16] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *AICM*, 4:389–396, 1995.
- [17] R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *The Third International Workshop on Implicit Surfaces*, pages 19–35. Eurographics, 1998.
- [18] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. In A. Glassner, editor, *SIGGRAPH '94 Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278. ACM SIGGRAPH, ACM Press, July 1994.
- [19] A. P. Witkin and J. M. Tenenbaum. On the role of structure in vision. In J. Beck, B. Hope, and A. Rosenfeld, editors, *Human and Machine Vision*, pages 481–543. Academic Press, New York, NY, 1983.

## **Implicit Modeling with PDE-based Techniques**

---

Haixia Du

Office of High Performance Computing and Communications

National Library of Medicine

National Institute of Health

## **Overview**

---

- Advantages of PDE-based techniques
- PDE formulation: from parametric to implicit
- Data and derivative constraints for implicit PDE
- Implicit PDE-based shape design, reconstruction and control
- Numerical methods to solve PDEs
- Conclusion and future work

## Partial Differential Equation (PDE)

- A PDE example
  - 2D Laplace's equation:  $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$
- PDE-based techniques
  - Consider differential property
  - Behavior of objects are governed by PDEs

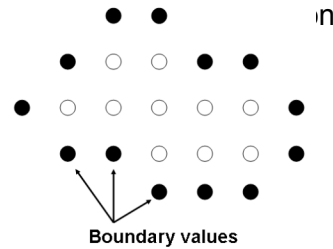
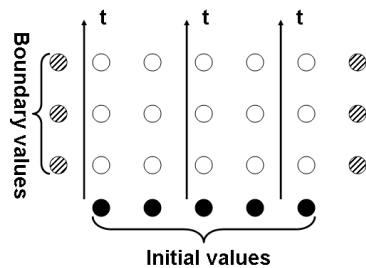
## PDE Classifications

$$A \frac{\partial^2 u}{\partial x^2} + 2B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D \frac{\partial u}{\partial x} + E \frac{\partial u}{\partial y} + Fu = G$$

- $B^2 - AC > 0$ : hyperbolic
  - Wave equation  $\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}$
- $B^2 - AC = 0$ : parabolic
  - Diffusion equation  $\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left( D \frac{\partial u}{\partial x} \right)$
- $B^2 - AC < 0$ : elliptic
  - Poisson equation  $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y)$

## PDE Classifications

- Initial value problem
  - Given information at  $t_0$ , the solution will propagate forward in time
- Boundary value problem
  - Given boundary information
  - Solution will be a static



## PDE Classifications

PDE types	Hyperbolic PDE	Parabolic PDE	Elliptic PDE
Initial value problem	Wave equation	Diffusion equation	
Boundary value problem			Poisson equation



## PDE-based Techniques for Graphics

---

- Image processing

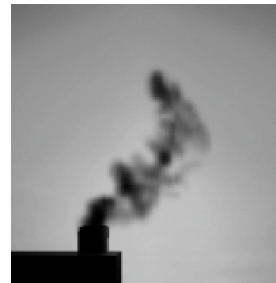
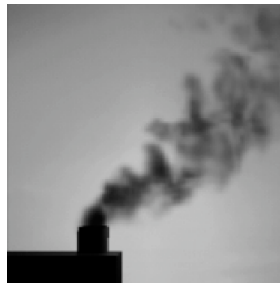
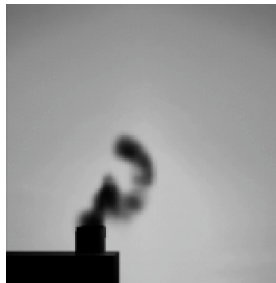


[Bertalmio *et al.* 00]

## PDE-based Techniques for Graphics

---

- Image processing
- Simulation and animation

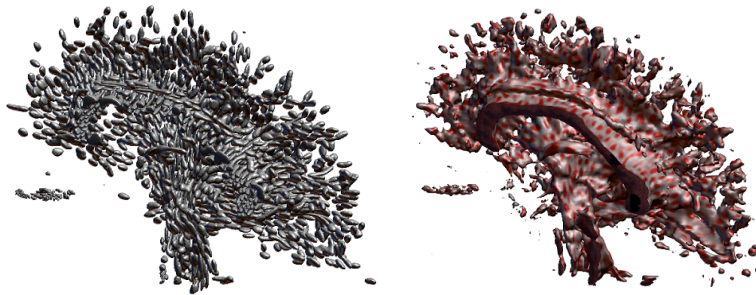


[Foster and Metaxas 97]

## PDE-based Techniques for Graphics

---

- Image processing
- Simulation and animation
- Visualization

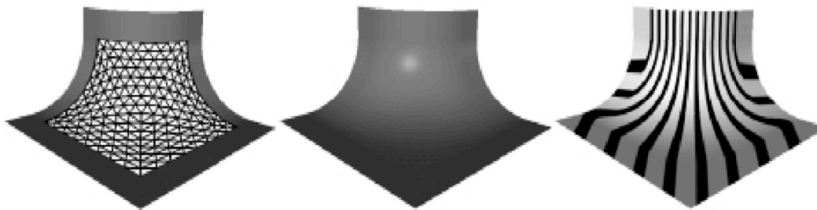


[Kindlmann *et al.* 00]

## PDE-based Techniques for Graphics

---

- Image processing
- Simulation and animation
- Visualization
- Geometric modeling



[Schneider and Kobbelt 00]

## Advantages of PDE-based Techniques

---

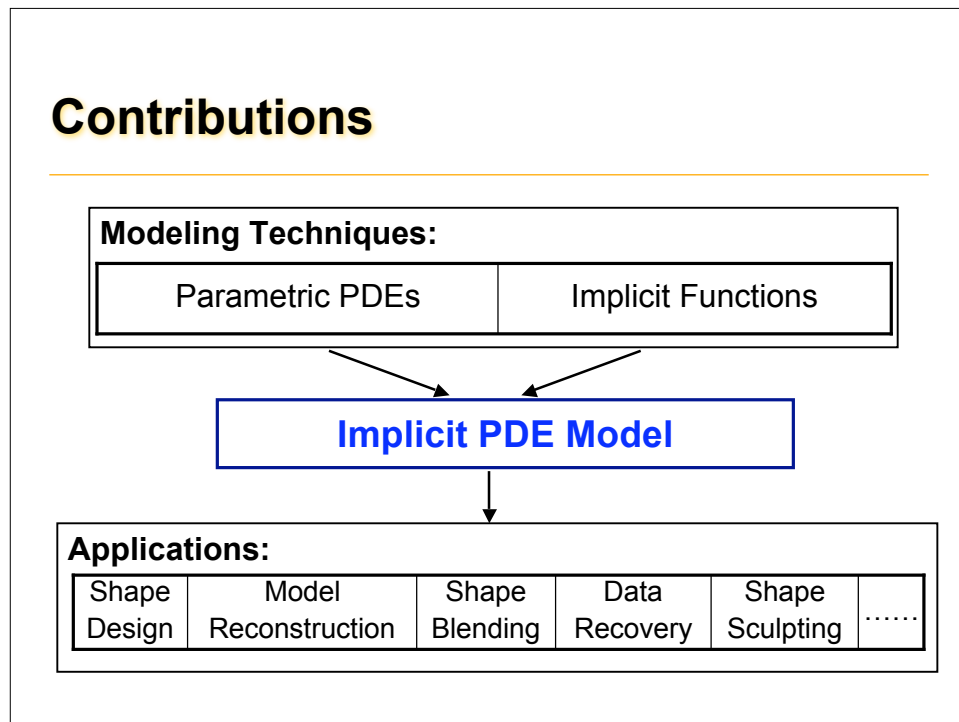
- Formulate natural physical process
- Satisfy continuity requirements
- Minimize energy functionals
- Use boundary/initial information without additional specifications
- Provide intuitive and natural control
- Unify geometric and physical attributes

## Motivation

---

- Maximizing modeling potentials of boundary-value PDEs and implicit functions
- General boundary constraints for various applications of implicit models
- Global and local control and manipulation for implicit shape modeling

## Contributions



## Related Work: PDE Surfaces and Solids

- Blending problem [Bloor and Wilson 89]
- Free-form surfaces [Bloor and Wilson 90]
- PDE solids [Bloor and Wilson 93]
- Interactive design [Ugail *et al.* 99]
- Physics-based interactive and direct surface sculpting [Du and Qin 00a, 00b, 05]
- PDE-based free-form deformation [Du and Qin 01]
- .....

## Parametric PDE Surfaces

- PDE surface formulation

$$\left( \frac{\partial^2}{\partial u^2} + a^2(u, v) \frac{\partial^2}{\partial v^2} \right)^2 \mathbf{X}(u, v) = 0$$

$$\mathbf{X}(u, v) = [x(u, v) \quad y(u, v) \quad z(u, v)]^T$$

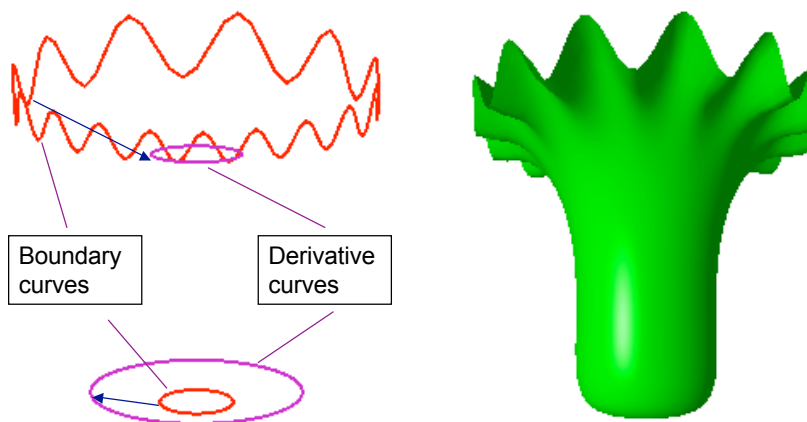
*Biharmonic equation if blending coefficient  $a(u, v) = 1$*

- Boundary conditions

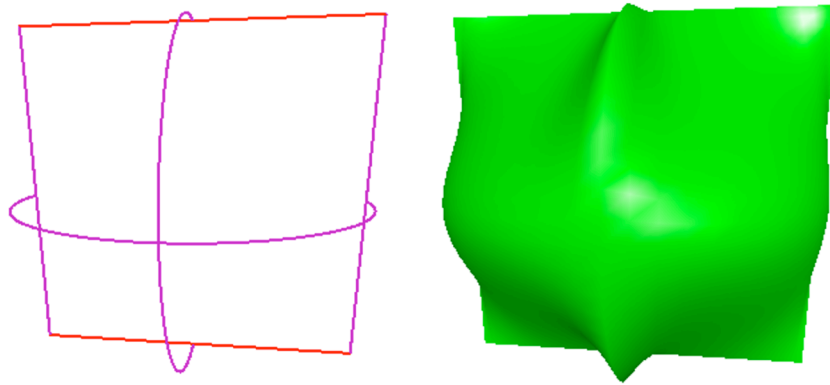
$$\mathbf{X}(u_i, v) = \mathbf{f}_i(v); \mathbf{X}(u, v_j) = \mathbf{g}_j(u)$$

$$\partial \mathbf{X}(u_i, v) / \partial u = \mathbf{s}_i(v); \partial \mathbf{X}(u, v_j) / \partial v = \mathbf{t}_j(u)$$

## Parametric PDE Surfaces



## Parametric PDE Surfaces

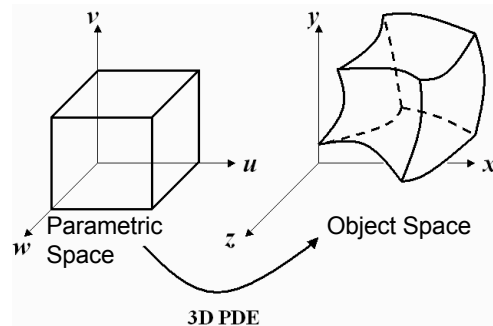


## Parametric PDE Solids

- PDE solid formulation

$$\left( a^2(u, v, w) \frac{\partial^2}{\partial u^2} + b^2(u, v, w) \frac{\partial^2}{\partial v^2} + c^2(u, v, w) \frac{\partial^2}{\partial w^2} \right)^2 \mathbf{X}(u, v, w) = 0$$

- Free-form deformation



## Parametric PDE Solids

- Boundary conditions:

- Surfaces:

$$\mathbf{X}(0, v, w) = \mathbf{U}_0(v, w), \mathbf{X}(1, v, w) = \mathbf{U}_1(v, w),$$

$$\mathbf{X}(u, 0, w) = \mathbf{V}_0(u, w), \mathbf{X}(u, 1, w) = \mathbf{V}_1(u, w),$$

$$\mathbf{X}(u, v, 0) = \mathbf{W}_0(u, v), \mathbf{X}(u, v, 1) = \mathbf{W}_1(u, v)$$

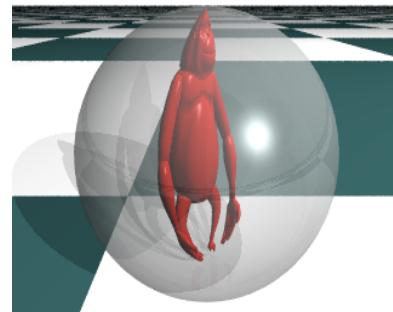
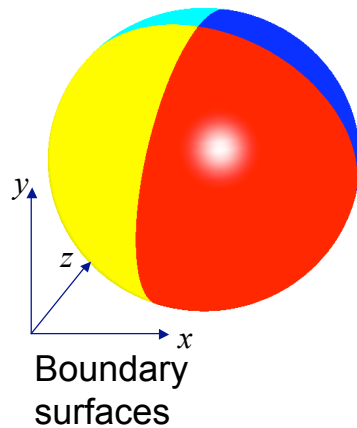
- Curve network:

$$\mathbf{X}(u, v_i, w_j) = \mathbf{U}_{ij}(u), v_i \in \{0, 1\} \text{ or } w_j \in \{0, 1\};$$

$$\mathbf{X}(u_k, v, w_l) = \mathbf{V}_{kl}(v), u_k \in \{0, 1\} \text{ or } w_l \in \{0, 1\};$$

$$\mathbf{X}(u_r, v_s, w) = \mathbf{W}_{rs}(w), u_r \in \{0, 1\} \text{ or } v_s \in \{0, 1\}$$

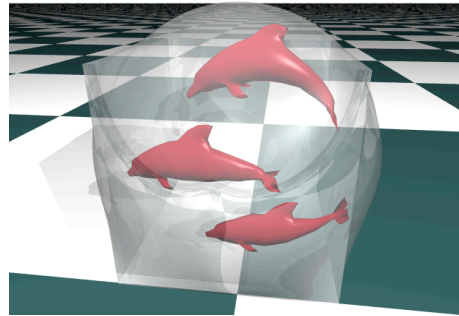
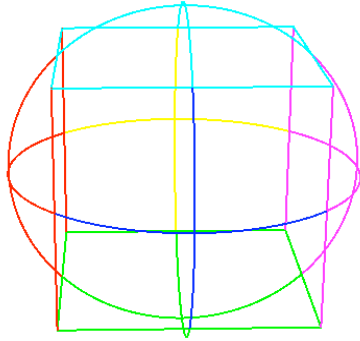
## PDE Solid from Boundary Surfaces



Corresponding PDE solid

## PDE Solid from Boundary Curves

---



## Implicit PDE Model

---

- Implicit PDE formulations:

$$\left( a^2(x, y, z) \frac{\partial^2}{\partial x^2} + b^2(x, y, z) \frac{\partial^2}{\partial y^2} + c^2(x, y, z) \frac{\partial^2}{\partial z^2} \right)^2 d(x, y, z) = 0$$

$$\left( a^2(x, y, z) \frac{\partial^2}{\partial x^2} + b^2(x, y, z) \frac{\partial^2}{\partial y^2} + c^2(x, y, z) \frac{\partial^2}{\partial z^2} \right) d(x, y, z) = 0$$

- $d(x, y, z)$ : the intensity function in the 3D physical space
- Not level-set method
- Model the entire implicit working space



## Boundary Constraints

---

- Generalized boundary constraints  
Boundary information, volumetric data, sketch curves, unorganized scattered data points
- Initialization of working space  
Finite-difference method (FDM), radial basis function (RBF) interpolation, distance field approximation, ...
- Smoothing and manipulation with FDM

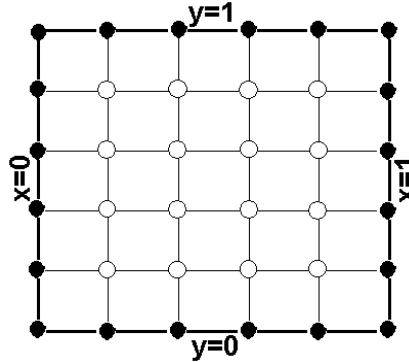
## Boundary Constraints

---

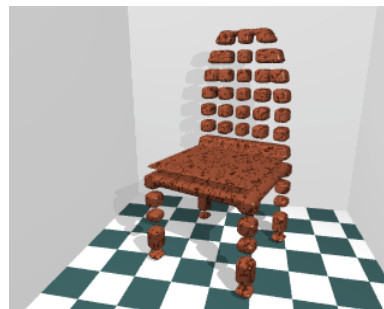
- Categorized based on applications
  - Traditional boundary conditions (cross-sectional constraints)
  - Boundary constraints for shape blending
  - Arbitrary sketch curves for implicit shape design
  - Unorganized scattered data points for model reconstruction

## Traditional Boundary Conditions

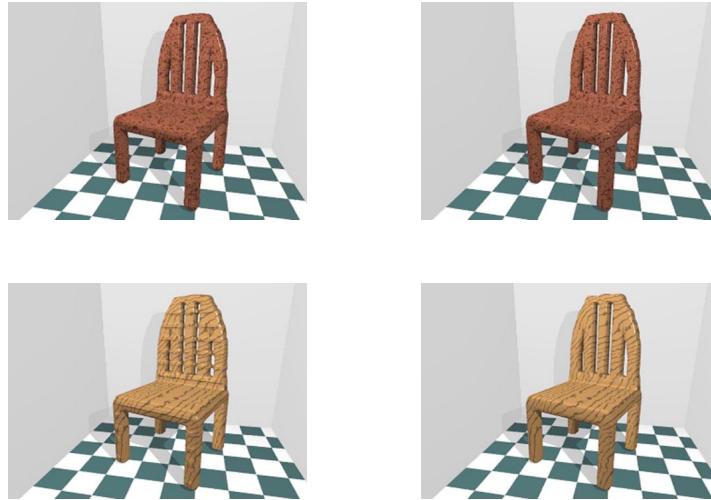
- Intensity values at boundaries
- Cross-sectional slices (optional)
- Can be solved using iterative method for FDM directly



## Traditional Boundary Conditions

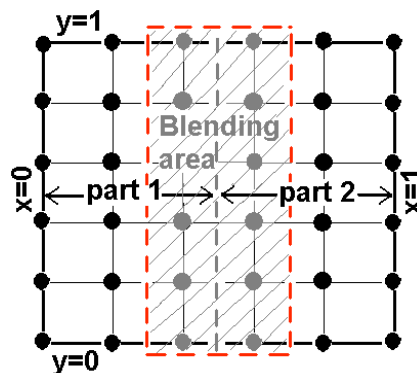


## Traditional Boundary Conditions

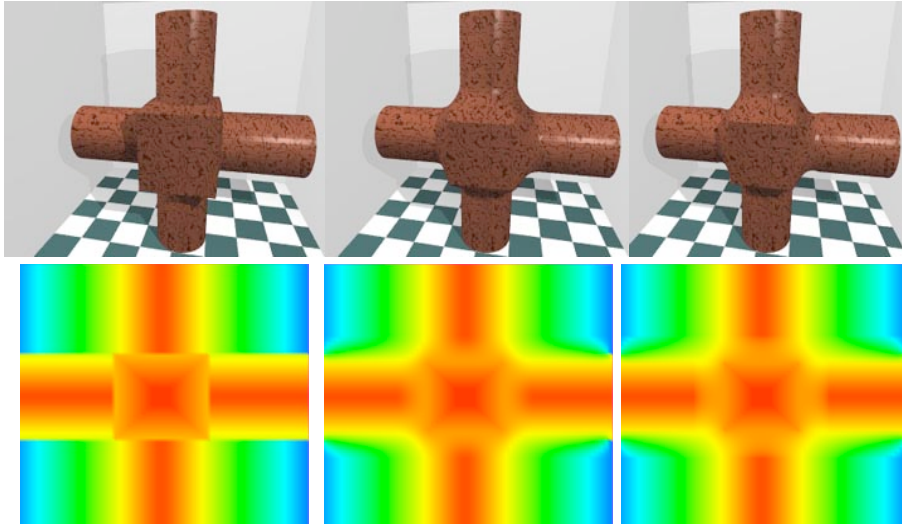


## Boundary Constraints for Blending

- Shapes to be blended in the working space
- Most information are given
- Only blended parts need to be computed
- Solve: FDM

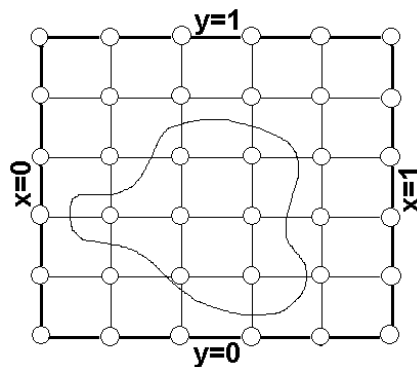


## Boundary Constraints for Blending



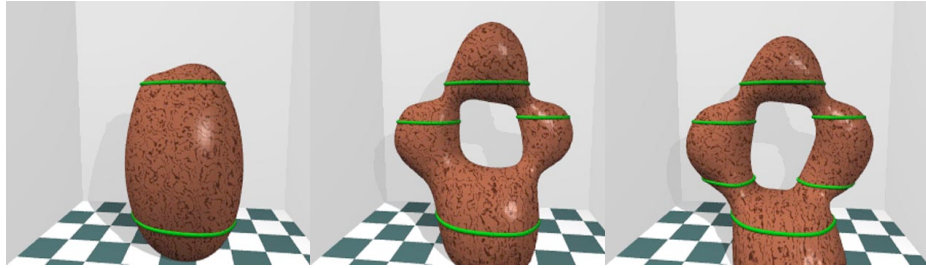
## Sketch Curve Constraints

- A set of 3D sketch curves to define implicit shape
- Boundary values of working space are unknown
- Initialization: RBF method
- Manipulation: FDM



## Sketch Curve Constraints

---

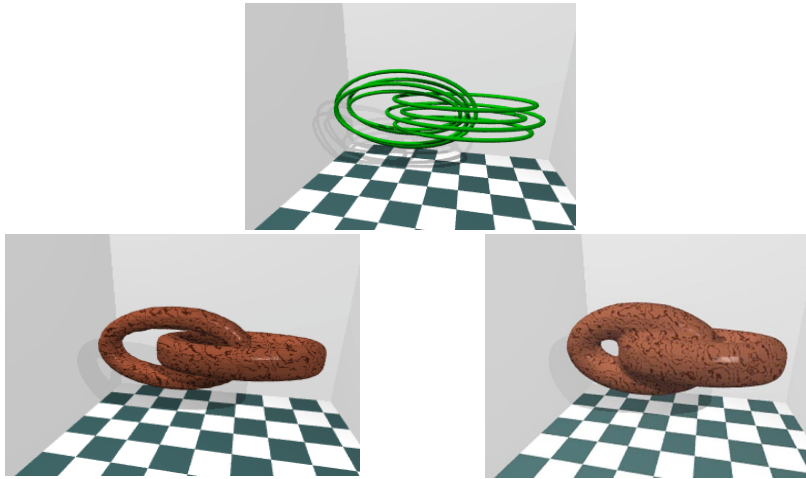


## Local RBF Method

---

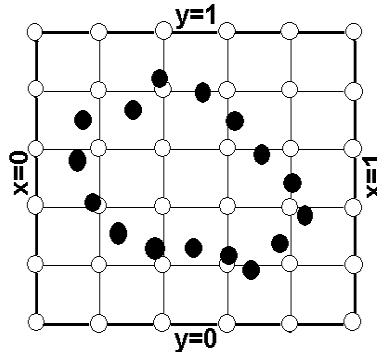
- RBF method defines global shape
  - Difficult for complex models
  - Compactly supported RBF calculates only a band around the data set
- Solution:
  - Perform RBF method at local regions
  - FDM for smoothing the entire working space
  - Arbitrary implicit shape blending

## Local RBF Method

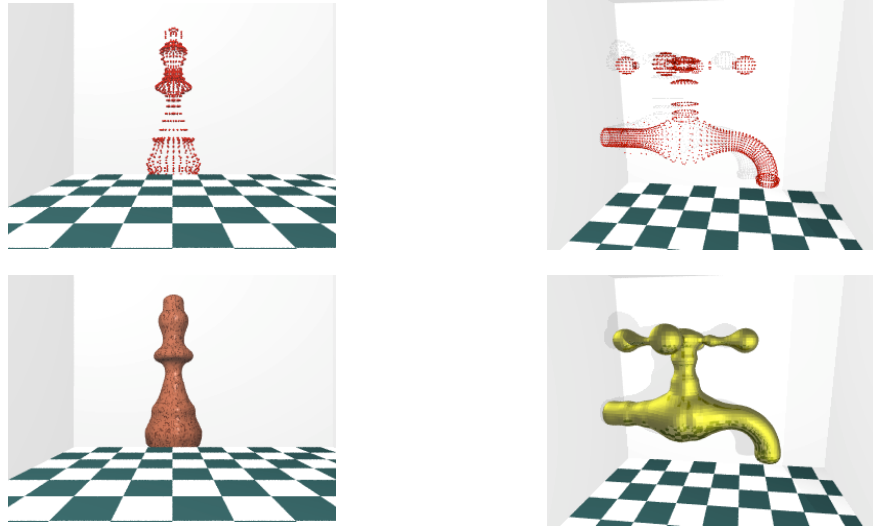


## Scattered Point Constraints

- Define shapes by a set of unorganized points
- Boundary values of working space are missing
- Initialization: distance field approximation (tagging algorithm [Zhao et.al. 01])
- Manipulation: FDM



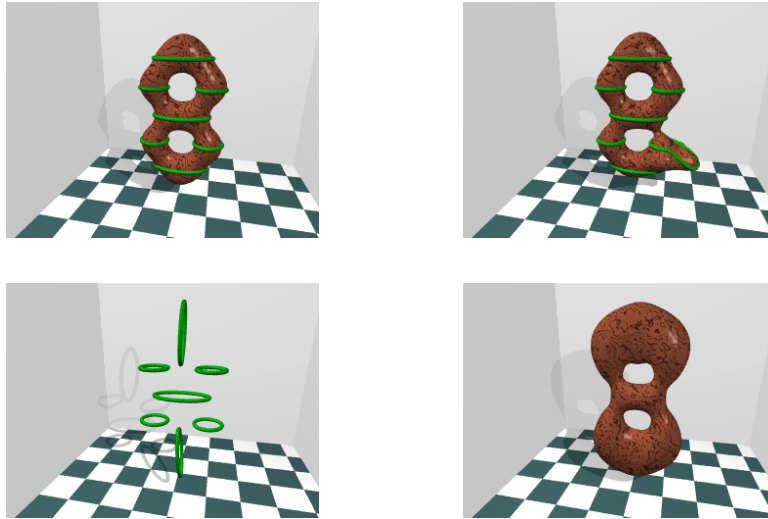
## Scattered Point Constraints



## Manipulation of Implicit PDE Objects

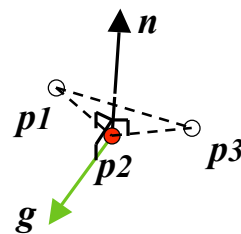
- Sketch curve sculpting
  - Shape, intensity, and gradient directions
- Blending coefficient manipulation
- Direct manipulations
  - Iso-contour deformation
  - Regional modification
  - CSG tools
  - Gradient sculpting
  - Curvature manipulation

## Sculpting of Sketch Curves



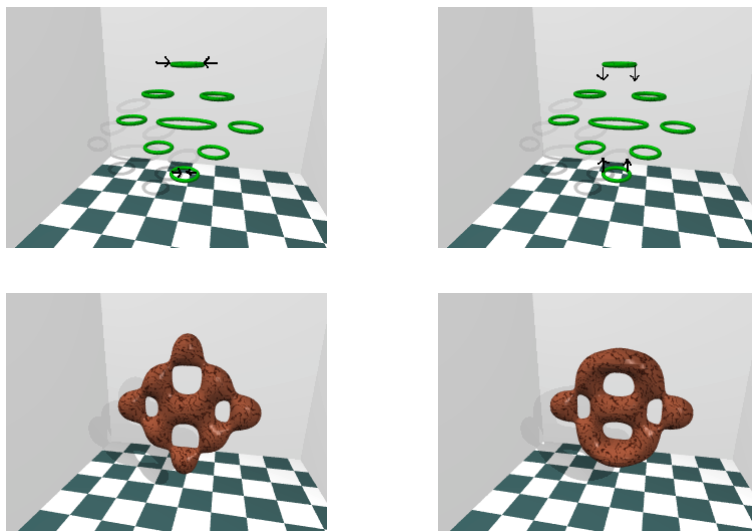
## Gradient Information of Sketch Curves

- For RBF initialization
- Gradient direction defines the increasing and decreasing directions of intensity distributions
- Changing gradient direction will change the implicit shape

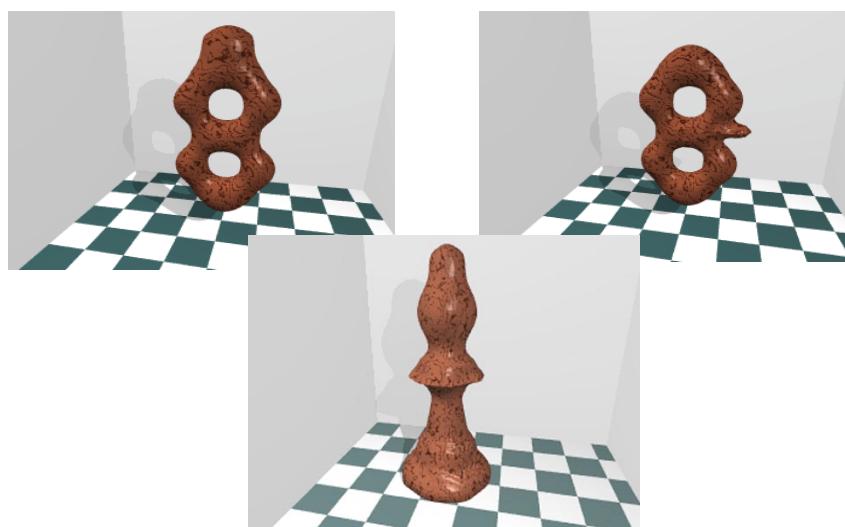




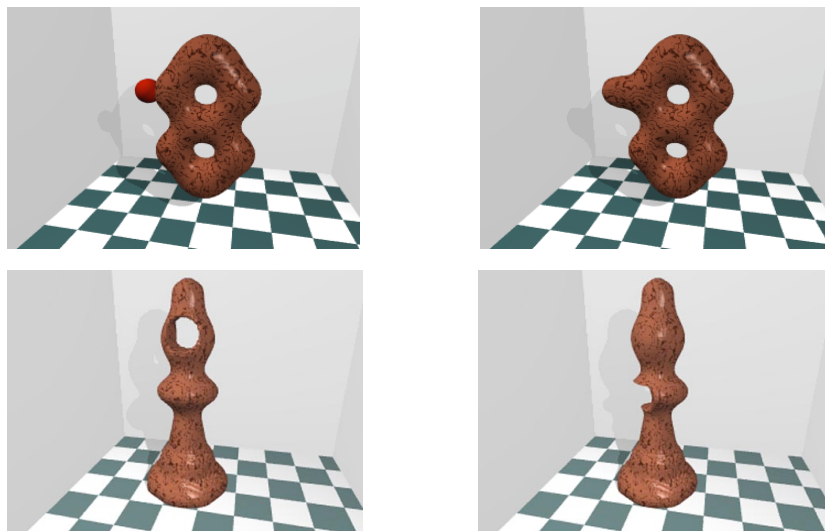
## Changing Gradient Directions



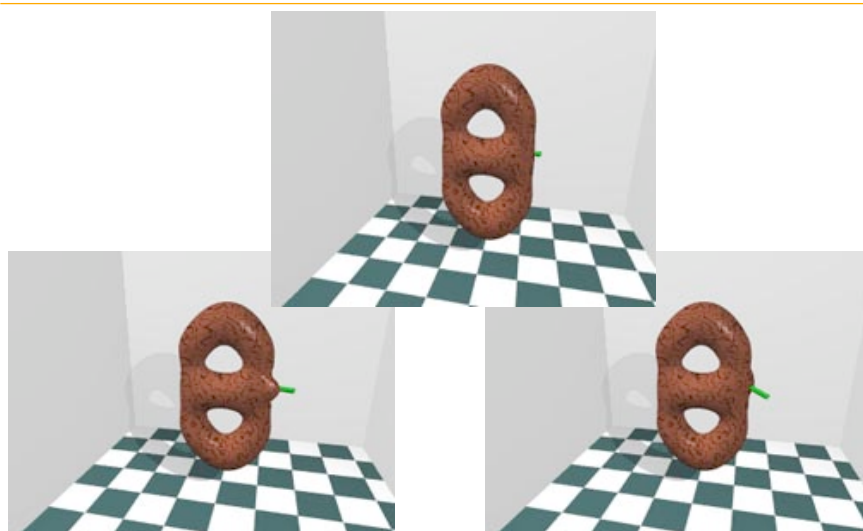
## Direct Intensity Sculpting



## Direct CSG Manipulations

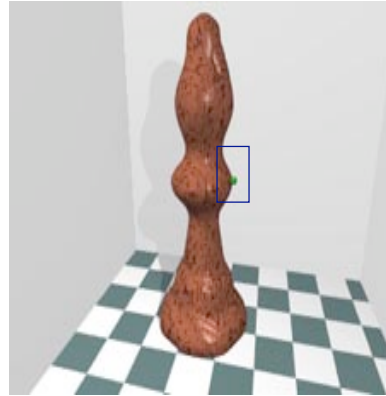
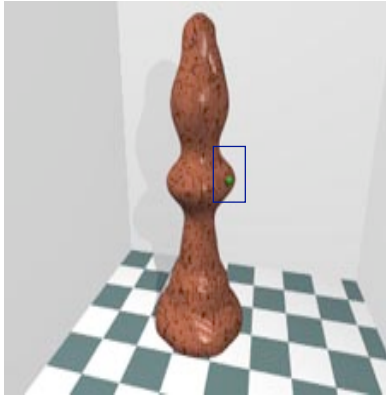


## Implicit Gradient Sculpting



## Curvature Manipulations

---



## Numerical Techniques

---

- Spectral approximation
- Finite-element method (FEM)
- Finite-difference method (FDM)

## Spectral Approximation

---

- Finite sum of closed-form functions + remainder
- Globally defined function
- Infinitely differentiable
- A closed band subject to periodic boundary conditions for 2D parametric PDE
- Separation of variables

## Spectral Approximation

---

$$\mathbf{X}(u, v) = \mathbf{F}(u, v) + \mathbf{R}(u, v)$$

$$\mathbf{F}(u, v) = \mathbf{A}_0(u) + \sum_{n=1}^N [\mathbf{A}_n(u) \cos(nv) + \mathbf{B}_n(u) \sin(nv)]$$

$$\mathbf{R}(u, v) = \mathbf{r}_1(v) e^{wu} + \mathbf{r}_2(v) u e^{wu} + \mathbf{r}_3(v) e^{-wu} + \mathbf{r}_4(v) u e^{-wu}$$

$$\mathbf{A}_0(u) = \mathbf{a}_{00} + \mathbf{a}_{01}u + \mathbf{a}_{02}u^2 + \mathbf{a}_{03}u^3$$

$$\mathbf{A}_n(u) = \mathbf{a}_{n1}e^{anu} + \mathbf{a}_{n2}ue^{anu} + \mathbf{a}_{n3}u^2e^{-anu} + \mathbf{a}_{n4}u^3e^{-anu}$$

$$\mathbf{B}_n(u) = \mathbf{b}_{n1}e^{anu} + \mathbf{b}_{n2}ue^{anu} + \mathbf{b}_{n3}u^2e^{-anu} + \mathbf{b}_{n4}u^3e^{-anu}$$

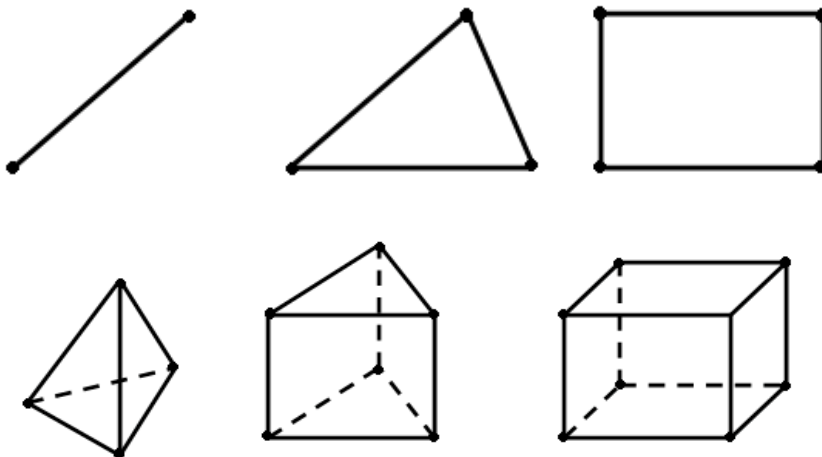
## Finite Element Method

---

- Approximate the infinite problem by interpolation functions over sub-domains
  - Discretize the domain into sub-domains
  - Select the interpolation functions
  - Formulate the system of equations
  - Solve the equations for coefficients of the interpolation to approximate the solution

## Typical Finite Elements

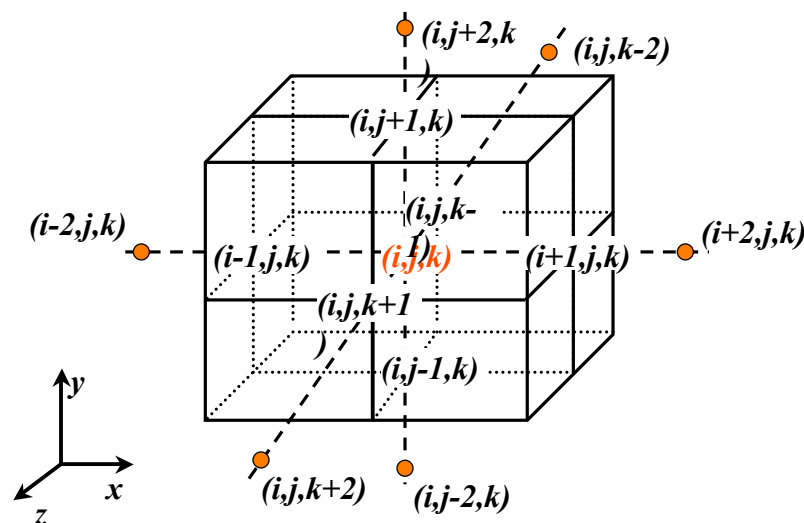
---



## Finite Difference Method

- Divides the working space into discrete grids
- Samples the PDE at grid points with discretized approximations
- Forms a set of algebraic equations
- Uses iterative techniques and multi-grid algorithm to improve the performance

## Working Space Discretization



## Finite Difference Approximations

$$\begin{aligned}\frac{\partial^4 d_{i,j,k}}{\partial x^4} &\approx \frac{d_{i-2,j,k} + d_{i+2,j,k} - 4d_{i-1,j,k} - 4d_{i+1,j,k} + 6d_{i,j,k}}{\Delta x^4}, \\ \frac{\partial^4 d_{i,j,k}}{\partial y^4} &\approx \frac{d_{i,j-2,k} + d_{i,j+2,k} - 4d_{i,j-1,k} - 4d_{i,j+1,k} + 6d_{i,j,k}}{\Delta y^4}, \\ \frac{\partial^4 d_{i,j,k}}{\partial z^4} &\approx \frac{d_{i,j,k-2} + d_{i,j,k+2} - 4d_{i,j,k-1} - 4d_{i,j,k+1} + 6d_{i,j,k}}{\Delta z^4}, \\ \frac{\partial^4 d_{i,j,k}}{\partial x^2 \partial y^2} &\approx \frac{d_{i-1,j-1,k} + d_{i+1,j-1,k} + d_{i-1,j+1,k} + d_{i+1,j+1,k} - 2(d_{i-1,j,k} + d_{i+1,j,k} + d_{i,j-1,k} + d_{i,j+1,k}) + 4d_{i,j,k}}{\Delta x^2 \Delta y^2}, \\ \frac{\partial^4 d_{i,j,k}}{\partial x^2 \partial z^2} &\approx \frac{d_{i-1,j,k-1} + d_{i+1,j,k-1} + d_{i-1,j,k+1} + d_{i+1,j,k+1} - 2(d_{i-1,j,k} + d_{i+1,j,k} + d_{i,j,k-1} + d_{i,j,k+1}) + 4d_{i,j,k}}{\Delta x^2 \Delta z^2}, \\ \frac{\partial^4 d_{i,j,k}}{\partial y^2 \partial z^2} &\approx \frac{d_{i,j-1,k-1} + d_{i,j+1,k-1} + d_{i,j-1,k+1} + d_{i,j+1,k+1} - 2(d_{i,j-1,k} + d_{i,j+1,k} + d_{i,j,k-1} + d_{i,j,k+1}) + 4d_{i,j,k}}{\Delta y^2 \Delta z^2}.\end{aligned}$$

## Difference Equations

$$\mathbf{AD} = \mathbf{b},$$

$$\mathbf{D} = \begin{bmatrix} d_{0,0,0} & d_{0,0,1} & \dots & d_{l-1,m-1,n-2} & d_{l-1,m-1,n-1} \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} A_{0,0,0} & A_{0,0,1} & \dots & A_{l-1,m-1,n-2} & A_{l-1,m-1,n-1} \end{bmatrix},$$

$$\begin{cases} A_{(i,j,k)(i,j,k)} = 1, b_{i,j,k} = c, & (i,j,k) \text{ is a constraint point;} \\ A_{i,j,k} \mathbf{D} = (a_{i,j,k}^2 \frac{\partial^2}{\partial x^2} + b_{i,j,k}^2 \frac{\partial^2}{\partial y^2} + c_{i,j,k}^2 \frac{\partial^2}{\partial z^2})^2 d_{i,j,k}, & \text{otherwise.} \end{cases}$$

- Constrained system:  $\mathbf{A}_c \mathbf{D} = \mathbf{b}_c$ 
  - Enforcing additional constraints by replacing the original equations by constraints

## Solving Difference Equations

---

- Iterative methods
  - Gauss-Seidel iteration
 
$$\mathbf{A}\mathbf{D} = \mathbf{b}, \mathbf{A} = \mathbf{A}_d - \mathbf{A}_r, \mathbf{A}_d\mathbf{D} = \mathbf{A}_r\mathbf{D} + \mathbf{b}$$

$$\mathbf{A}_d\mathbf{D}^{(n)} = \mathbf{A}_r\mathbf{D}^{(n-1)} + \mathbf{b}$$
  - Successive over-relaxation (SOR) iteration
- Multi-grid method improvement
  - Starting from coarsest grids
  - Linearly interpolating the coarse solution to get initial guess of finer resolution

## Finite Difference Method

---

- Simple and easy for implementation
- Allows flexible and generalized boundary conditions and additional constraints
- Enables local control and direct manipulation
- Guarantees an approximate solution
- Time performance depends on resolution of discretization of working space
- Possible improvement by parallel computing



## Conclusion

---

- Unify Implicit functions and boundary-value PDE models
- General constraints for shape design, reconstruction, blending, and recovery
- Global and local implicit shape modeling with numerical techniques
- Intuitive and interactive sculpting toolkits
  - Sketch curve sculpting, direct manipulation of implicit objects, blending coefficient control

## Future Work

---

- Hierarchical structure to model features and details
- More efficient solver for implicit PDE
- Implicit PDE-based shape transformation
- Manipulation tools with haptics

.....



ELSEVIER

Computer-Aided Design 36 (2004) 1101–1116

COMPUTER-AIDED  
DESIGN

[www.elsevier.com/locate/cad](http://www.elsevier.com/locate/cad)

# A shape design system using volumetric implicit PDEs

Haixia Du\*, Hong Qin

*Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY, USA*

Received in revised form 12 October 2003; accepted 9 January 2004

## Abstract

Solid modeling based on partial differential equations (PDEs) can potentially unify both geometric constraints and functional requirements within a single design framework to model real-world objects via its explicit, direct integration with parametric geometry. In contrast, implicit functions indirectly define geometric objects as the level-set of underlying scalar fields. To maximize the modeling potential of PDE-based methodology, in this paper we tightly couple PDEs with volumetric implicit functions in order to achieve interactive, intuitive shape representation, manipulation, and deformation. In particular, the unified approach can reconstruct the PDE geometry of arbitrary topology from scattered data points or a set of sketch curves. We make use of elliptic PDEs for boundary value problems to define the volumetric implicit function. The proposed implicit PDE model has the capability to reconstruct a complete solid model from partial information and facilitates the direct manipulation of underlying volumetric datasets via sketch curves and iso-surface sculpting, deformation of arbitrary interior regions, as well as a set of CSG operations inside the working space. The prototype system that we have developed allows designers to interactively sketch the curve outlines of the object, define intensity values and gradient directions, and specify interpolatory points in the 3D working space. The governing implicit PDE treats these constraints as generalized boundary conditions to determine the unknown scalar intensity values over the entire working space. The implicit shape is reconstructed with specified intensity value accordingly and can be deformed using a set of sculpting toolkits. We use the finite-difference discretization and variational interpolating approach with the localized iterative solver for the numerical integration of our PDEs in order to accommodate the diversity of generalized boundary and additional constraints.

© 2004 Elsevier Ltd. All rights reserved.

**Keywords:** Partial differential equation techniques; Implicit functions; Volume graphics; Shape design; Geometric constraints; Scattered data fitting

## 1. Introduction

Partial differential equation (PDE) techniques are widely used for many visual computing applications, such as nature phenomena simulation and animation [17], variational fairing [35], image inpainting [2], etc. They also provide an alternative way for geometric design [5–7,45]. Different from traditional geometric representations, the PDE methods model graphical objects as solutions of certain elliptic PDEs with boundary constraints inside the parametric domain. The parametric PDE model simplifies the geometric design process by using only boundary conditions to recover the whole interior information and offers high-order continuity as well as energy minimization properties. However, it is extremely difficult to model arbitrary shapes

of general topology, because the PDEs are defined over regular parametric domain, like traditional parametric approaches.

In contrast, implicit functions use level-sets of certain scalar field functions in the physical domain directly to design, model, and interact with 3D objects, without constructing the mapping between parametric and physical spaces. They offer a fundamentally different yet convenient and natural design paradigm (in comparison with parametric representations) in visual computing fields such as graphics, animation, and geometric design. This is because of their unique properties such as arbitrary topology, collision detection, free of parametric correspondence, etc. Applications of implicit functions include shape blending, surface reconstruction from scattered data points, shape transformation, and interactive modeling [3,4,8–10,12,13,18,21,22,26,27,29,30,32,36,39,40,43,44,47].

Implicit functions offer several modeling advantages such as flexible topology, simple data structure, efficient

\* Corresponding author.

E-mail addresses: [dhaixia@cs.sunysb.edu](mailto:dhaixia@cs.sunysb.edu) (H. Du); [qin@cs.sunysb.edu](mailto:qin@cs.sunysb.edu) (H. Qin).

storage, volumetric information, unbounded geometry, etc. Nonetheless, most of implicit functions focus on surface models. Previous techniques for interactive implicit volume sculpting have certain modeling limitations. Recently, Cutler et al. [11] presented a procedural framework for specifying layered solid models and applying a series of simulation operations (serving as sculpting tools described by a script language) to complex models. Bærentzen and Christensen [1] developed an interactive volume sculpting system using level-set method. Museth et al. [28] proposed level-set-based editors for CSG operations, blending, embossing, and smoothing for implicit surfaces. However, these tools are associated with the specification of speed functions for the evolving level-set, which are non-intuitive for common users. Turk and O'Brien [41] presented interactive implicit surface sculpting via particles, but each operation requires reformatting and recalculation of the entire system, which is difficult to model large datasets. In general, the modeling potential of implicit functions has not been fully explored yet and there are still difficulties to design, reconstruct, and sculpt implicit models directly and intuitively.

To maximize the modeling capabilities of PDE techniques and implicit functions in geometric and visual computing areas, we propose a more general PDE-based modeling paradigm which integrates the PDE techniques with implicit functions into one single framework for interactive shape design and manipulation on PDE-based volumetric implicit models. We develop an implicit modeling system governed by elliptic PDEs of scalar intensity fields. In particular, our prototype system can reconstruct implicit objects and the embedding implicit 3D working space as approximated solutions of the PDEs by specifying a set of curve outlines or scattered data points of certain intensity values as *general* boundary constraints with the assistance of variational interpolating approaches. Because the curves and datasets are not required to be closed, open surfaces can be modeled within our system. Moreover, it offers a set of sculpting toolkits to manipulate implicit objects, such as interactively modifying the geometric shape, intensity value, and gradient direction of selected sketch curves, directly changing intensity values of selected regions in the working space, as well as deforming iso-contours at specified intensity values of the objects. Because the working space is governed by the PDEs, any missing information inside the space can be recovered by solving the PDEs according to the given constraints. Our system is able to recover damaged datasets using partial information, smooth the intensity distribution of volume data, and smoothly blend objects inside the working space. In general, our system allows intensity manipulations at any iso-value anywhere in the implicit working space to model implicit objects either directly or indirectly, which offers users both local and global control of the implicit PDE model.

This implicit PDE approach has modeling advantages of both parametric PDE techniques and implicit functions. First, the behavior of the implicit PDE model is governed by differential equations. Solving the PDEs results in both boundary and interior information simultaneously, which offers an alternative way to model implicit objects by using only boundary information. This property makes the PDE method extremely suitable for shape blending process. Second, many natural physical processes are characterized by differential equations in principle [19,20,37]. Hence, PDE models are natural and close to the real world. They are potentially ideal candidates for design, simulation, and analysis tasks. Furthermore, geometric objects with high-order continuity requirements can be readily defined through high-order PDEs because of their differential properties. Third, smooth objects that minimize certain energy functionals are the solutions of differential equations from the variational analysis point of view, so optimization techniques can be unified with PDE models. In addition, because the implicit PDE is formulated on a scalar intensity field and defines objects by collecting points of certain iso-values, it is capable of designing arbitrary topological shapes and recovering the full information from partial input, which reduces the burden of specifying the large quantity of constraints for complete datasets. It offers users a natural way to design objects easily with general non-isoparametric arbitrary curve outlines, reconstruct objects from scattered data points, blend shapes in the working space, and recover damaged datasets.

The remainder of the paper is structured as follows. Section 2 reviews the related work of PDE techniques and implicit models. We detail the PDE formulation and present our integrated approach for implicit PDE objects in Section 3. We introduce possible applications of our implicit PDE model by enforcing different types of boundary and additional constraints in Section 4. Section 5 discusses techniques of directly manipulating implicit PDE objects with constraints to construct more flexible topological shapes, such as sketch sculpting and local region manipulations. We outline the system implementation in Section 6.

## 2. Related work

Different from traditional free-form spline-based modeling techniques, Bloor and Wilson [5] introduced a method that defines smooth surfaces as solutions of *elliptic* PDEs. Since its initial application on surface blending, the PDE approach has broadened its applications for free-form surface design, solid modeling, and interactive surface editing [6,7,42] during the past decade. In principle, the PDE-based method has the advantage that most of the information defining an object comes from its boundaries. This permits an object to be generated and controlled by a very few parameters such as boundary-value conditions and global coefficients associated with an elliptic PDE. This

PDE technique was then used for modeling parametric surfaces and solids with global geometric features. To obtain interactive sculpting and local control, we [14,15] proposed an integrated model which combined PDE surfaces and physics-based modeling techniques to offer users direct manipulation for the PDE surfaces with generalized boundary constraints and user-specified features. We [16] extended the PDE technique's coverage from surfaces to solids in order to provide users a set of direct editing toolkits to model the real-world objects with interior material distribution. Zhang and You [45] investigated three different orders, i.e. second, mixed, and fourth-order of PDEs as surface representation techniques and demonstrated the use and effectiveness of the PDE method for free-form surface design.

However, because the aforementioned PDE methods define objects over the regular parametric domain, they (like other parametric representation techniques) have limitations in handling arbitrary topological shapes, which can be easily achieved by implicit functions.

Implicit functions offer a different way for shape modeling by using certain scalar field functions to define geometric entities. In the past several years, implicit functions have been widely developed as a powerful design and manipulation tool for graphical models. In 1994, Witkin and Heckbert [44] introduced an approach using particles to sample and control implicit surfaces. A set of particles are locked onto a surface and act as *control points* for the implicit surface. The surface shape can be manipulated by moving particles interactively. Ferley et al. [18] presented a *sculpture metaphor* for rapid shape prototyping. In their approach, the sculpted shape is defined as the iso-surface of a spatially sampled scalar field and can be manipulated by adding, removing, painting, or smoothing material and applying free-form and stamp tools. These techniques only provide interactive and practical sculpting tools for implicit surfaces.

As for implicit solids, Savchenko et al. [34] introduced a novel approach for the reconstruction of geometric models from given point sets using volume splines. Raviv and Elber [32] presented an interactive sculpting technique that uses the zero level-set of the scalar, tensor-product, uniform trivariate B-spline functions to represent 3D objects. The trivariate functions have a control volume that consists of scalar control coefficients. Users can indirectly sculpt the object by modifying relevant scalar control coefficients of the trivariate B-spline functions in different levels of details. Hua and Qin [23,24] developed interactive solid sculpting toolkits with haptics on implicit B-spline solids defined through the use of B-spline control coefficients over the intensity field. However, the control of B-spline coefficients is less intuitive to ordinary users in general.

Implicit functions can also be used for shape reconstruction and 3D morphing process. Turk and O'Brien [40] used variational implicit functions to achieve shape morphing and surface reconstruction. They employed the radial basis

function (RBF) method to construct an implicit function, which interpolates the given dataset and minimizes the thin-plate energy. Yet since the RBF method is a global variational interpolating approach, any changes in the dataset will cause recalculation of the entire system. It's time-consuming for direct manipulation and not applicable for local sculpting of complex implicit models. Ohtake et al. [29] presented a *multi-level partition of unity* implicit surface supporting local features, but it is sensitive to the quality of input data.

Level-set method is another popular technique to model implicit objects. Zhao et al. [47] proposed a *weighted* minimal surface model based on variational formulations and PDE techniques to construct a surface from scattered data. They used the level-set method as a numerical technique to evolve the implicit surface continuously following the gradient descent of the energy functional for the final reconstruction. Their level-set model is governed by a time evolution PDE with velocity at the level-sets given by the motion equation of the original surface. The level-set method is based on a continuous formulation using PDEs and deforms an implicit surface according to various equations of motion depending on geometry, external forces, or certain energy minimization. It can easily handle topological changes and reduce noises in the dataset. The level-set method mainly focuses on implicit objects reconstructed from scattered datasets. Problems for interpolating curve sketches, especially open curve sketches have not been addressed. The shape deformation using the level-set method is often obtained by manipulating the speed functions in the level-set formulation [1,28], which is non-intuitive for general users.

Despite the modeling advantages of implicit functions, there are still difficulties for intuitive design and direct manipulation of implicit surfaces and solids in general. We integrate the implicit functions with the parametric PDE to offer users modeling advantages of both types of techniques. Instead of time evolution PDEs used in the level-set method, we employ static elliptic PDEs for boundary value problems. In particular, we introduce a novel technique which defines volumetric implicit objects as solutions of the elliptic PDEs of scalar intensity fields under *generalized* boundary constraints, including sketch curves, scattered data points, as well as volumetric datasets. The constraints may be associated by different intensity values, which offers more degrees of freedom than previous implicit techniques. Our implicit PDE model can be used for geometric shape design, object reconstruction, damaged data recovery, and shape blending. Implicit PDE objects can be manipulated by modifying the initial constraints or directly changing intensity values in the interior of the volumetric space. The implicit PDE method recovers not only the target object, but also the entire working space by the given information. Our system does not require the constraints to be closed datasets, which provides modeling potentials for open surfaces. To visualize implicit objects of scalar

intensity field, we can either use the Marching Cube method [25] which calculates the triangulated iso-surface at a selected intensity value on discretized sampling grids, or output the volumetric data in the working space to other volume rendering systems such as Pov-Ray or Vol-Vis systems.

### 3. Formulating implicit PDEs

#### 3.1. Implicit elliptic PDE formulation

The implicit PDEs employed in this paper are founded upon the parametric PDE solid models [16]. In order to take advantage of the interactive feature associated with the parametric PDE modeling techniques, we use elliptic PDEs to define scalar intensity field for modeling implicit objects. Because higher-order PDEs can provide higher-order continuity for the scalar intensity value distribution, we employ a fourth-order elliptic PDE to model the scalar field for smooth results with tangential continuity, especially when dealing with shape blending and damage data recovery in which most of information are specified as constraints.

In particular, we formulate the unknown function as the intensity field function  $d(x, y, z)$  defined in the 3D physical space of  $x, y$ , and  $z$ . The corresponding implicit PDE is formulated as follows:

$$\left( a^2 \frac{\partial^2}{\partial x^2} + b^2 \frac{\partial^2}{\partial y^2} + c^2 \frac{\partial^2}{\partial z^2} \right)^2 d(x, y, z) = 0, \quad (1)$$

where  $x, y$ , and  $z$  are coordinate variables of 3D physical space varying from 0 to 1, respectively, which form a unit cube as the working space;  $a, b$ , and  $c$  are arbitrary blending coefficient functions of  $x, y, z$  defining material properties of the implicit space, which are initially defined as constants throughout the entire working space. The blending coefficient functions control the relative intensity blending and the level of variable dependence among  $x, y$ , and  $z$  directions. For example, according to Eq. (1), if  $a$  is given as a large value for all sampling points in the working space, then the contributions of  $d(x, y, z)$  along  $x$  direction will be relatively small in comparison with the other two directions. Hence, the coefficient functions will affect the solution of Eq. (1).

Because the numerical techniques used in this paper to solve the fourth-order elliptic PDE are suitable for other boundary value PDEs, we also incorporate a second-order PDE into our system:

$$\left( a^2 \frac{\partial^2}{\partial x^2} + b^2 \frac{\partial^2}{\partial y^2} + c^2 \frac{\partial^2}{\partial z^2} \right) d(x, y, z) = 0, \quad (2)$$

which is less time-consuming to solve with less continuous intensity distribution and can be used for initial guess of intensity values of the objects.

Because  $a(x, y, z), b(x, y, z)$ , and  $c(x, y, z)$  are allowed to vary across  $d(x, y, z)$ , i.e. different locations in the physical domain may have different smoothing coefficient values, local control on implicit PDE objects can be easily achieved.

To obtain direct and local manipulations on the implicit PDE objects, we solve Eqs. (1) and (2) using numerical methods based on finite-difference approximations of the PDEs, which require at least six boundary conditions at  $x = 0, x = 1, y = 0, y = 1, z = 0, z = 1$  defining the intensity values at three boundary surface pairs of the 3D working space in order to derive a unique solution. However, in most applications, there are no such boundary conditions available for modeling implicit objects, especially in the case of using implicit functions for shape reconstruction, where the constraints are usually defined by certain contouring sketch curves or scattered points assigned with specified intensity values inside the 3D working space. In such cases, the intensity distributions on the boundaries are unknown. Thus, such problems cannot be solved by traditional finite-difference methods directly. However, we may approximate the intensity distribution for this type of problems as follows. First, we find an initial guess of the volumetric working space using certain techniques. Second, we use the guessed boundary values as boundary conditions, and enforce additional constraints according to the original data. Third, we perform iterative finite-difference techniques to get an approximated solution for the entire working space based on these constraints. After that, direct manipulations and local sculpting inside the working space can be enforced by adding additional constraints to the PDEs. Variational interpolating approaches are good candidates for shape reconstruction from scattered points, such as the RBF method [26,40] which creates a 3D implicit function to give an approximation interpolating the given constraints by minimizing certain energy functionals. We employ the RBF method to compute the initial guess of the implicit PDE objects defined by sketch curves. We can also calculate the intensity values on sampling grids using their distance to the constraints, because the implicit objects can be defined by distance functions. The algorithm we use to compute the distance field is the fast-tagging approach proposed by Zhao et al. [46]. Note that, because our goal is to obtain an initial guess of the working space according to constraints, there are other techniques which can provide satisfactory results.

#### 3.2. Radial basis function

RBF is commonly used for scattered data interpolation, which is to generate a smooth surface that passes through a given set of scattered points. Scattered data interpolation sometimes can also be addressed using variational analysis where the desired solution is a function,  $f(\vec{x})$ , which minimizes certain energy functionals. In principle, the energy functional measures the quality of interpolation



subject to the interpolatory constraints  $f(\vec{c}_i) = h_i$ . It can be solved by a weighted sum of certain RBFs (note that, we use  $\phi(\vec{x}) = |\vec{x}|^3$  in this paper). Then the interpolation function can be formulated as:

$$f(\vec{x}) = \sum_{i=1}^n w_i \phi(\vec{x} - \vec{c}_i) + P(\vec{x}), \quad (3)$$

where  $\vec{c}_i$ 's are coordinate vectors of the constraints,  $w_i$ 's are weights, and  $P(\vec{x})$  is a polynomial only consisting of the linear and constant portions of  $f$ . According to the properties of the appropriate RBFs, the interpolation function minimizes the thin-plate energy while satisfying the data interpolation requirement. By applying the constraints to Eq. (3), we can obtain a linear equation system whose unknowns are the weights and coefficients of the polynomial  $P$ . This system can be solved using standard solvers of linear equations.

However, the RBF method requires gradient information of the datasets, and time and space complexity of the equation system depends on the number of constraints, so it is not suitable for reconstruction and interactive sculpting of large scattered datasets with arbitrary constraints. Because our goal here is to simply make an initial guess for our implicit PDE shape, distance approximation techniques such as fast-tagging algorithm which computes the signed distance field of the working space according to the constraints can give satisfactory results for such input.

### 3.3. Numerical simulation

In order to easily enforce additional constraints for direct manipulations of implicit objects, we resort to numerical techniques based on the finite-difference approximation and iterative methods for linear equations to solve the implicit PDEs with predefined boundary values or approximated initial guess from sketch curves/scattered points. The iterative methods will arrive at an approximated solution with user-specified error tolerances. Numerical algorithms also facilitate the material modeling of anisotropic distribution. A multi-grid-like iterative solver is used to improve the system performance.

The finite-difference method divides the working space into discrete grids along  $x, y, z$  directions and transforms a continuous PDE into a set of simultaneous algebraic equations by sampling the partial derivatives in the equation for each grid point with their finite-difference approximations. The algebraic equation system can be solved numerically either through a direct procedure or an iterative process for an approximated solution of the continuous PDE.

Based on Taylor's expansion, the derivatives of a univariate function can be approximated using the central-difference scheme  $f'(x) = (f(x+h) - f(x-h))/2h$ ,  $f''(x) = [f(x+h) - 2f(x) + f(x-h)]/h^2$ , where  $h$  denotes the spatial interval along  $x$  direction. This can be generalized to all partial derivatives on trivariate implicit geometry,

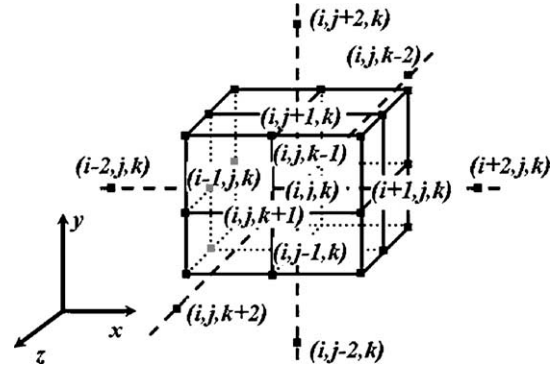


Fig. 1. The point discretization of an implicit function.

by dividing  $x, y, z$  domain into  $l, m$ , and  $n$  discretized grids, respectively. We represent the function  $d(x, y, z)$  by its values at the discrete set of points  $(x_i = i\Delta x, y_j = j\Delta y, z_k = k\Delta z)$ ,  $i = 0, 1, \dots, l-1$ ,  $j = 0, 1, \dots, m-1$ , and  $k = 0, 1, \dots, n-1$ .  $\Delta x, \Delta y, \Delta z$  are the grid spacing along  $x, y, z$  directions. We write  $d_{i,j,k}$  for  $d(x_i, y_j, z_k)$  and  $\{i, j, k\}$  for grid point  $(x_i, y_j, z_k)$  for sake of simplicity (Fig. 1). We use finite-difference representations of second-order and fourth-order partial derivatives  $(\partial^2 d_{i,j,k})/(\partial x^2)$ ,  $(\partial^4 d_{i,j,k})/(\partial x^4)$ , and  $(\partial^4 d_{i,j,k})/(\partial x^2 \partial y^2)$  at  $\{i, j, k\}$  as examples:

$$\begin{aligned} \frac{\partial^2 d_{i,j,k}}{\partial x^2} &= \frac{d_{i-1,j,k} + d_{i+1,j,k} - 2d_{i,j,k}}{(\Delta x)^2}, \\ \frac{\partial^4 d_{i,j,k}}{\partial x^4} &= \frac{d_{i-2,j,k} + d_{i+2,j,k} - 4d_{i-1,j,k} - 4d_{i+1,j,k} + 6d_{i,j,k}}{(\Delta x)^4}, \\ \frac{\partial^4 d_{i,j,k}}{\partial x^2 \partial y^2} &= \frac{d_{i-1,j-1,k} + d_{i-1,j+1,k} + d_{i+1,j-1,k} + d_{i+1,j+1,k}}{(\Delta x)^2 (\Delta y)^2} \\ &\quad + \frac{-2d_{i,j,k} - 2d_{i+1,j,k} - 2d_{i,j-1,k} - 2d_{i,j+1,k} + 4d_{i,j,k}}{(\Delta x)^2 (\Delta y)^2}. \end{aligned}$$

Other partial derivatives along  $y$  and  $z$  directions can be computed similarly.

Substituting partial derivatives by finite-difference representations at grid points, Eq. (1) can be rewritten as:

$$\mathbf{A}\mathbf{D} = \mathbf{b}, \quad (4)$$

where  $\mathbf{A}$  represents the discretized differential operator in  $(l \times m \times n) \times (l \times m \times n)$  matrix form, and each row in  $\mathbf{A}$  consists of coefficients of the difference equation for the corresponding grid point.  $\mathbf{A}$  is also controlled by the blending functions  $a(x, y, z)$ ,  $b(x, y, z)$ , and  $c(x, y, z)$ .  $\mathbf{D}$  collects the unknown intensity values at the grid points, and  $\mathbf{b}$  is defined by the value of constraints:

$$\begin{aligned} \mathbf{A} &= [\mathbf{A}_{(0,0,0)}, \mathbf{A}_{(0,0,1)}, \dots, \mathbf{A}_{(l-1,m-1,n-1)}]^T, \\ \mathbf{A}_{(i,j,k)} &= [A_{(i,j,k),(0,0,0)}, \dots, A_{(i,j,k),(l-1,m-1,n-1)}], \\ \mathbf{D} &= [d_{(0,0,0)}, d_{(0,0,1)}, \dots, d_{(l-1,m-1,n-1)}]^T, \\ \mathbf{b} &= [b_{(0,0,0)}, b_{(0,0,1)}, \dots, b_{(l-1,m-1,n-1)}]^T. \end{aligned}$$

$\mathbf{A}$  and  $\mathbf{b}$  are defined as follows: given a grid point  $\{i, j, k\}$ , let its index  $d = i \times l \times m + j \times m + k$  be represented as  $(i, j, k)$ . If it is a constraint point, all elements in  $\mathbf{A}_{(i,j,k)}$  have value 0 except  $A_{(i,j,k),(i,j,k)} = 1$ , and  $b_{(i,j,k)}$  is set to be the intensity value defined by the constraint. If it is free, the value of  $A_{(i,j,k),(i',j',k')}$  depends on contribution of  $\{i', j', k'\}$  in the difference equation at  $\{i, j, k\}$ , and  $b_{(i,j,k)} = 0$ . Fig. 1 shows the grid points contributing for  $\{i, j, k\}$  in the  $d$ th row of  $\mathbf{A}$ , i.e.  $\mathbf{A}_{(i,j,k)}$ . All the values of  $A_{(i,j,k),(i',j',k')}$  are set to be 0 except:

$$\begin{aligned} A_{(i,j,k),(i,j,k)} &= 6 \left( \frac{a_{i,j,k}^4}{\Delta x^4} + \frac{b_{i,j,k}^4}{\Delta y^4} + \frac{c_{i,j,k}^4}{\Delta z^4} \right) \\ &\quad + 8 \left( \frac{a_{i,j,k}^2 b_{i,j,k}^2}{\Delta x^2 \Delta y^2} + \frac{a_{i,j,k}^2 c_{i,j,k}^2}{\Delta x^2 \Delta z^2} + \frac{b_{i,j,k}^2 c_{i,j,k}^2}{\Delta y^2 \Delta z^2} \right), \\ A_{(i,j,k),(i \pm 1, j, k)} &= -4 \left( \frac{a_{i,j,k}^4}{\Delta x^4} + \frac{a_{i,j,k}^2 b_{i,j,k}^2}{\Delta x^2 \Delta y^2} + \frac{a_{i,j,k}^2 c_{i,j,k}^2}{\Delta x^2 \Delta z^2} \right), \\ A_{(i,j,k),(i, j \pm 1, k)} &= -4 \left( \frac{b_{i,j,k}^4}{\Delta y^4} + \frac{a_{i,j,k}^2 b_{i,j,k}^2}{\Delta x^2 \Delta y^2} + \frac{b_{i,j,k}^2 c_{i,j,k}^2}{\Delta y^2 \Delta z^2} \right), \\ A_{(i,j,k),(i, j, k \pm 1)} &= -4 \left( \frac{c_{i,j,k}^4}{\Delta z^4} + \frac{a_{i,j,k}^2 c_{i,j,k}^2}{\Delta x^2 \Delta z^2} + \frac{b_{i,j,k}^2 c_{i,j,k}^2}{\Delta y^2 \Delta z^2} \right), \\ A_{(i,j,k),(i \pm 2, j, k)} &= \frac{a_{i,j,k}^4}{\Delta x^4}, \\ A_{(i,j,k),(i, j \pm 2, k)} &= \frac{b_{i,j,k}^4}{\Delta y^4}, \\ A_{(i,j,k),(i, j, k \pm 2)} &= \frac{c_{i,j,k}^4}{\Delta z^4}, \\ A_{(i,j,k),(i \pm 1, j \pm 1, k)} &= \frac{2a_{i,j,k}^2 b_{i,j,k}^2}{\Delta x^2 \Delta y^2}, \\ A_{(i,j,k),(i \pm 1, j, k \pm 1)} &= \frac{2a_{i,j,k}^2 c_{i,j,k}^2}{\Delta x^2 \Delta z^2}, \\ A_{(i,j,k),(i, j \pm 1, k \pm 1)} &= \frac{2b_{i,j,k}^2 c_{i,j,k}^2}{\Delta y^2 \Delta z^2}. \end{aligned}$$

The matrix  $\mathbf{A}$  is called ‘tridiagonal with fringes’ [31].

Similarly, using finite-difference techniques, Eq. (2) can be rewritten as:

$$\mathbf{A}'\mathbf{D} = \mathbf{b}', \quad (5)$$

Our implicit PDE is open along all of  $x$ ,  $y$ , and  $z$  directions, so forward/backward difference approximations shall be utilized when computing partial derivatives near the six boundaries instead. Arbitrary boundary and additional constraints can be easily enforced by the finite-difference method. In our system, after making the initial guess of the intensity values, we fix the intensity values at boundaries, so that the manipulations on the implicit objects can be performed using the finite-difference iterative solver. In general, this type of elliptic PDEs

allows designers to choose (various) constraints based on diverse design tasks.

### 3.4. Constrained system

One attractive advantage of the PDE modeling techniques is that the interior of the objects is controlled by PDEs without the need of extra specification for interior material distribution. More importantly, users can modify an implicit PDE object by enforcing additional hard constraints of desired intensity values anywhere inside the working space without violating previously defined conditions. Additional hard constraints inside the working space introduce a set of new equations into the system to replace the corresponding original difference equations. For example, if we want to set the intensity value  $d_{i,j,k}$  as a particular constant value  $d_0$ , the equation  $d_{i,j,k} = d_0$  will be used to replace the discretized difference equation approximating the PDE at the point  $\{i, j, k\}$ , i.e.  $A_{(i,j,k),(i,j,k)} = 1$ , all other  $A_{(i,j,k),(i',j',k')} = 0$ , and  $b_{(i,j,k)} = d_0$ . After replacing all the equations according to the constraints, Eq. (4) becomes

$$\mathbf{A}_c \mathbf{D} = \mathbf{b}_c, \quad (6)$$

where  $\mathbf{A}_c$  and  $\mathbf{b}_c$  are obtained by replacing  $k(k > 0)$  equations in the original system with those derived from additional  $k$  constraints at the corresponding coordinate positions. The constrained system for the second-order Eq. (5) has the similar form:

$$\mathbf{A}'_c \mathbf{D} = \mathbf{b}'_c, \quad (7)$$

### 3.5. Iterative method

With boundary conditions, we solve the linear Eqs. (4)–(7) using finite-difference-based iterative techniques. These methods make immediate use of the sparse matrix structure on the left-hand side of the equations. Using the matrix  $\mathbf{A}$  in Eq. (4) as an example,  $\mathbf{A}$  is split into two parts

$$\mathbf{A} = \mathbf{A}_d - \mathbf{A}_r, \quad (8)$$

where  $\mathbf{A}_d$  consists of the diagonal elements of  $\mathbf{A}$  and zeros elsewhere, and  $\mathbf{A}_r$  is the remainder. Then Eq. (4) becomes

$$\mathbf{A}_d \mathbf{D} = \mathbf{A}_r \mathbf{D} + \mathbf{b}. \quad (9)$$

The iterative methods start from choosing an initial guess  $\mathbf{D}^{(0)}$  and then solving successively by iterating  $\mathbf{D}^{(s)}$  from

$$\mathbf{A}_d \mathbf{D}^{(s)} = \mathbf{A}_r \mathbf{D}^{(s-1)} + \mathbf{b}. \quad (10)$$

The same idea can be applied to Eqs. (5)–(7).

In the case of predefined boundary conditions, we compute the initial guess using simple linear interpolations based on the constraints. The iteration will stop at  $\mathbf{D}^{(s)}$  for an approximated solution when the difference between  $\mathbf{D}^{(s)}$  and

$D^{(s-1)}$  is less than a threshold (we use  $10^{-9}$  in this paper). Certain variants of iterative techniques exist for solving the aforementioned linear equations [38]. In this paper, we employ the Gauss-Seidel iteration, which uses the updated value of the iteration result at a grid point on the right-hand side of Eq. (10) as soon as it becomes available. To further speed up the converging rate of Gauss-Seidel iteration, we take into account the error factor, which is characterized by the difference between the approximation and the real solution. This leads to the method of successive over-relaxation iteration, or SOR iteration. Nonetheless, the discretization of volumetric implicit PDE space results in a very large number of linear equations. This causes the slow convergence of iterative methods. To achieve a solution faster, we start solving the equations at a coarse grid with down-sampled constraints and interpolate the solution at finer grids to compute the initial guess for the iterative methods at the finer resolution. The convergent rate of the iterative solvers can be greatly increased.

#### 4. Boundary conditions for different applications

To construct an implicit PDE object, first we need to outline the rough shape of the object, which can be defined through boundary conditions or special constraints such as curve contours and scattered data points in the working space that the object interpolates. The form of boundary

constraints varies for different applications. Our implicit PDE techniques accept boundary conditions for applications such as shape blending, object recovery, and shape reconstruction from sketch curves and scattered data points. Fig. 2 illustrates different types of boundary conditions in simplified 2D cases.

##### 4.1. Shape design using traditional boundary constraints

The implicit PDE techniques can model geometric shapes by computing the information of the whole working space based on traditional boundary constraints with optional cross-sectional details inside the working space. Such boundary conditions are defined as intensity values sampled at certain resolution from input or use some analytic functions to generate implicit boundary functions  $d(0, y, z)$ ,  $d(1, y, z)$ ,  $d(x, 0, z)$ ,  $d(x, 1, z)$ ,  $d(x, y, 0)$ ,  $d(x, y, 1)$  and a collection of cross-sectional scalar intensity functions  $d(x_i, y, z)$ ,  $d(x, y_j, z)$ , or  $d(x, y, z_k)$ , where  $x_i, y_j, z_k \in (0, 1)$  are constants. These functions are sampled at specified resolution to provide a set of intensity values inside the working space. Using these values as generalized boundary conditions, we introduce certain number of new equations and the linear equation system has the form of Eqs. (6) or (7) which can be solved using above mentioned techniques. Fig. 3 shows examples of the fourth-order and second-order PDEs, respectively. Although Eq. (6) takes more time to solve, it provides higher-order continuity of intensity

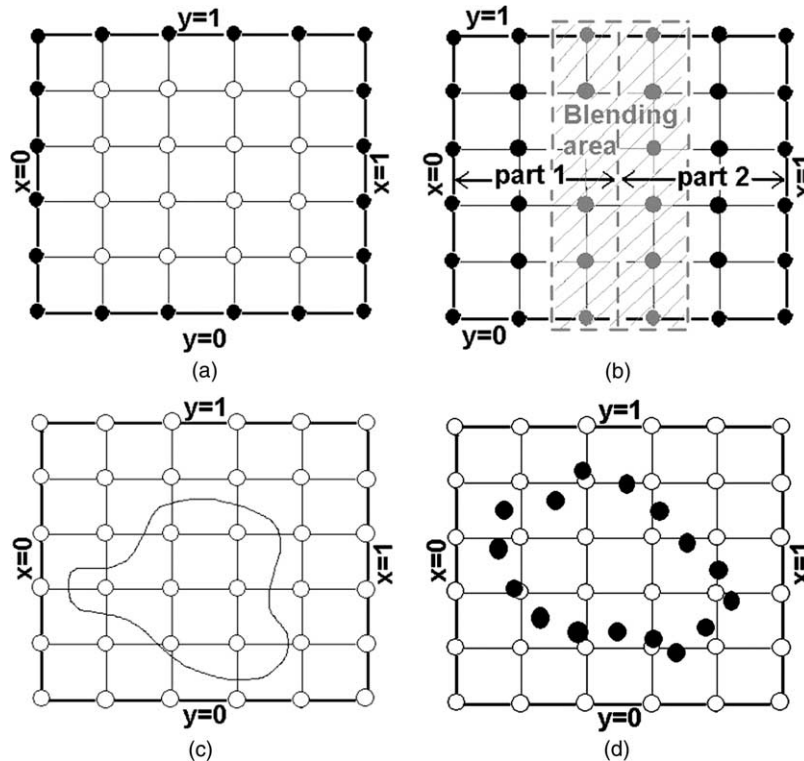


Fig. 2. 2D illustrations of different types of boundary conditions. (a) Traditional boundary constraints; (b) boundary conditions for shape blending; (c) sketch-curve constraints; (d) scattered-point constraints.



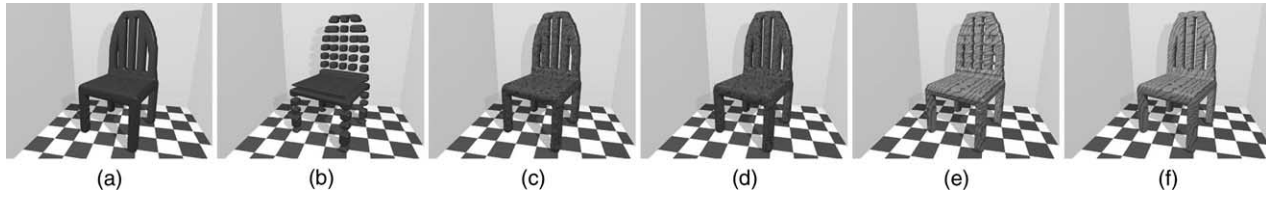


Fig. 3. Examples of implicit PDE objects generated from cross-sectional boundary conditions. (a) Original object rendered by POV-RAY; (b) cross-sectional boundary conditions by removing several data slices along the y-direction from the original data; (c) and (d) are recovered fourth-order implicit PDE objects from (b) by solving Eq. (1) with  $b = 1.2$  and  $4.8$ , respectively; (e) and (f) are corresponding second-order objects. The fourth-order PDE provides more continuous results.

distributions in the working space in comparison with the results from Eq. (7).

#### 4.2. Shape blending

Our PDE formulations define the interior information of implicit objects via differential properties, which means that it is possible to automatically recover the missing information from partial data using our prototype system and guarantee intensity continuity of non-constrained parts of the working space. This feature can be applied to shape blending process by placing the objects to be blended into the working space and the system will compute the connecting parts between those objects. Such kind of datasets form another type of initialization with pre-defined boundary constraints, which gives most of the information with only a small portion of the working space missing. The missing information of the working space can be approximated based on the remaining part using our PDE formulations. An example of shape blending is shown in Fig. 4 including blended results using different order PDEs, where the fourth-order blended shape is smoother than the second-order result. The above two types of boundary conditions allow our system to model volumetric datasets.

#### 4.3. Shape reconstruction from sketch curves

To maximize the modeling potential of implicit PDEs, we develop a set of toolkits using PDE techniques to reconstruct objects from spatial sketch curves of specified intensity values. Because with this type of constraints, the boundary information around the working space is missing, it is extremely difficult to directly solve the implicit PDE

under such constraints. Therefore, we employ techniques such as the RBF method for the interpolation problems to obtain an initial guess for the implicit PDE shapes subject to sketch curve constraints. We then use the iterative solver to get a smooth solution. When performing the RBF method, the gradient information indicating the change of the intensity values around the constraints will be needed to define the inside and the outside of the reconstructed shape. If the gradient information is not provided by users, our system calculates the gradient at each sample point of the constraints according to the normal of the local tangent plane of the curve at that point, as explained in Fig. 5. Our system also allows designers to interactively input certain sketch curves such as B-spline curves with specified intensity values, which permits the initial sketch curves being modified directly. Note that, the sketch curves are not required to be planar curves. Moreover, they can even be open curves, which may result in open iso-surfaces instead of solid objects. Fig. 6 shows examples obtained from sketch curves.

When modeling more complex shapes from sketches, usually there are a large number of sketch curves to be enforced, which will increase the number of calculations dramatically. Moreover, sometimes the sketch curves are only designed to model the local area they resides, so their global contribution are not desirable. To address such issues, our system allows users to compute the initial guess of implicit PDE objects using the RBF method for selected subset of sketch curves at any local region of the working domain without disturbing the outside areas. At the initialization stage, when using RBF method to compute the initial guess of the implicit shape, users are prompted to select interested curves, define the region in the working

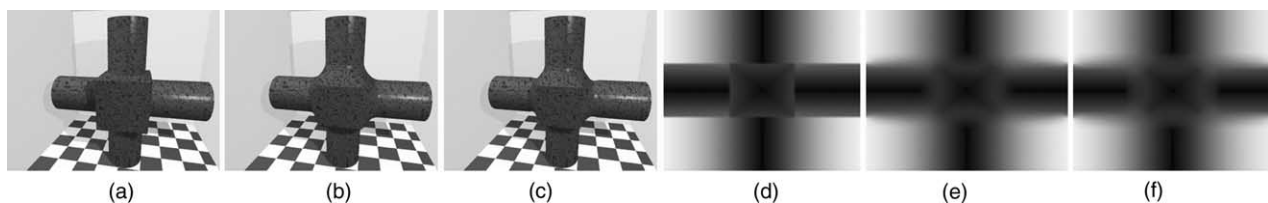


Fig. 4. Shape blending using implicit PDEs. (a) Original dataset shown in iso-surface; (b) blended object from (a) using the fourth-order PDE; (c) blended object using the second-order PDE; (d)–(f) are cross-section views of the working space for (a)–(c), respectively, where darkness increases with intensity.

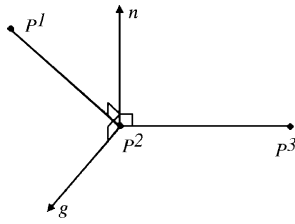


Fig. 5. Illustration of computing the gradient direction.  $p_1, p_2$ , and  $p_3$  are neighboring points on a discretized curve;  $\vec{n}$  is the normal of their local plane; and  $\vec{g}$  is the gradient vector for  $p_2$ .

space to reconstruct the subset of the object, as well as to indicate if curves that only part of them inside the specified area can make contribution to the reconstruction. After all the sampled intensity values in each of the sub-regions of the working space are computed, our system can perform a global blending process to put sub-regions together. This feature can reduce the number of calculations of the RBF method, and provide fast reconstruction by sculpting sketch curves. Moreover, CSG sculpting tools can be easily enforced accordingly. Fig. 7 shows an example.

#### 4.4. Shape reconstruction from unorganized scattered data points

Implicit functions are commonly used for shape reconstruction from scattered data points. In this paper, our implicit PDE model not only reconstructs objects from unorganized scattered data sets, but also recovers information of the entire working space where objects reside, with which direct manipulations of objects can be easily applied. Similar to the sketch curve constraints,

intensity values at boundaries of the working space are unknown. However, for scattered points datasets where the number of constraints is extremely large and there is no gradient information available, RBF method is not suitable for computing the initial guess. In such case, we use the signed distance field approximation based on the constraints. The initial intensity value on the sampling grids are computed by the fast-tagging algorithm introduced by Zhao et al. [46] based on their signed distance to the data point constraints and we then use iterative solvers to conduct a smoothing task. Two examples are shown in Fig. 8.

### 5. Sculpting and manipulation toolkits for implicit PDEs

Our system provides a set of toolkits for global deformation and local editing of the implicit objects. Fig. 9 shows a snapshot of our prototype system while manipulating a selected sketch curve.

#### 5.1. Modifying blending coefficients

The coefficient functions  $a(x, y, z)$ ,  $b(x, y, z)$ , and  $c(x, y, z)$  can influence the solution of the implicit PDEs. They control the relative intensity blending and the level of variable dependence among  $x$ ,  $y$ , and  $z$  directions, thus they can be treated as generalized material properties over the volumetric working space. Consequently, users can control how the boundary and additional conditions influence the interior intensity distribution by modifying the length scale at arbitrary locations (i.e.  $a_{i,j,k}$ ,  $b_{i,j,k}$ , and  $c_{i,j,k}$ ). In general, users can

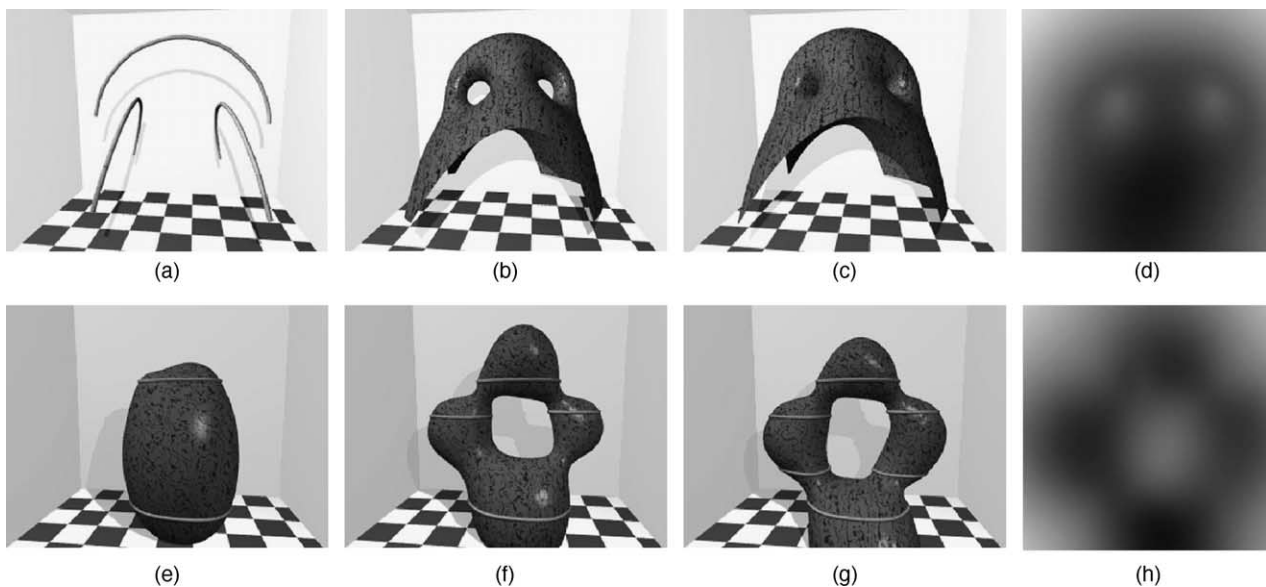


Fig. 6. Examples of shape reconstruction from sketch curves. (a) is a set of open curves without specified gradient information; (b) and (c) are iso-surfaces at different iso-values, respectively; (d) is a cross-section view of the implicit shape; (e)–(g) show an example of generating implicit shapes by incrementally defining a set of B-spline curves; (e) is an object defined by two curves; (f) is the refined object by adding two additional sketch curves; (g) is the shape reconstructed from six B-spline sketch curves; and (h) is a cross-section view.

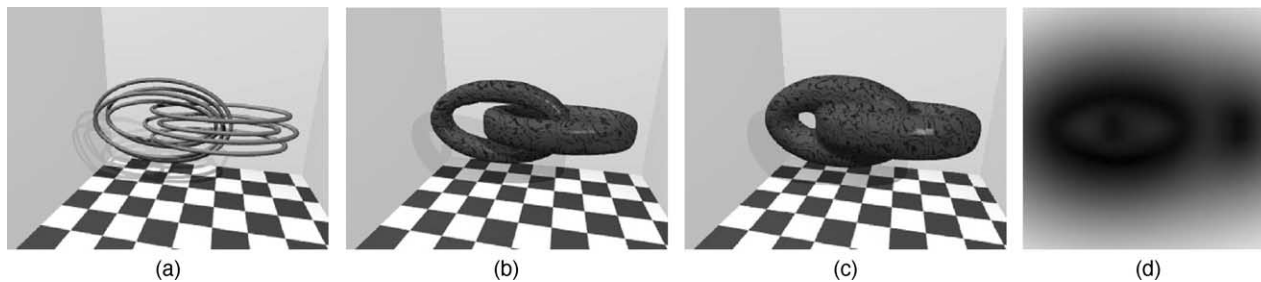


Fig. 7. Example for performing the RBF initialization locally. (a) Two set of sketch curves; (b) and (c) are reconstructed implicit shapes rendered at different iso-values; (d) a cross-section view.

define the control functions  $a(x, y, z)$ ,  $b(x, y, z)$ , and  $c(x, y, z)$  interactively over the specified grid point  $\{i, j, k\}$ . Our system allows users to modify them locally to deform the shape. Fig. 3 has examples of implicit PDE objects subject to different coefficient values.

### 5.2. Sketch curve sculpting

Implicit objects can be defined by specifying a set of sketch curves which outline the rough shape of the objects. Our implicit PDE model provides interactive shape design toolkits to allow users to manipulate the sketch curves in order to deform the underlying reconstructed implicit object. The sketch curves defining the rough shape of the object can be obtained by either predefined curve network or B-spline curves from users' direct input. Our system allows users to modify the geometric shape, intensity value, as well as gradient directions of the sketch curves interactively in order to get the desired object.

In order to modify the sketch curves smoothly, B-spline approximations for those curves are calculated at the initialization stage, then users can sculpt the curves interactively by manipulating the B-spline control points via sculpting, translation, and rotation. Because the reconstructed implicit object is required to interpolate those sketch curves, which define its outlining shape approximately, it will follow the shape changes accordingly. Fig. 11 has an example of sculpting the shape of a selected sketch curve. The intensity values of sketch curves decide where the final shape of the implicit objects should pass through at the level-set of its value. By modifying the intensity values of selected curves, users can manipulate the objects accordingly. Furthermore, according to the gradient definition, the intensity values increase along gradient directions of sketch curves and decrease in the opposite directions in general. Gradient directions provide information of the intensity distributions starting at the sketch curves and propagating to the neighborhood, which

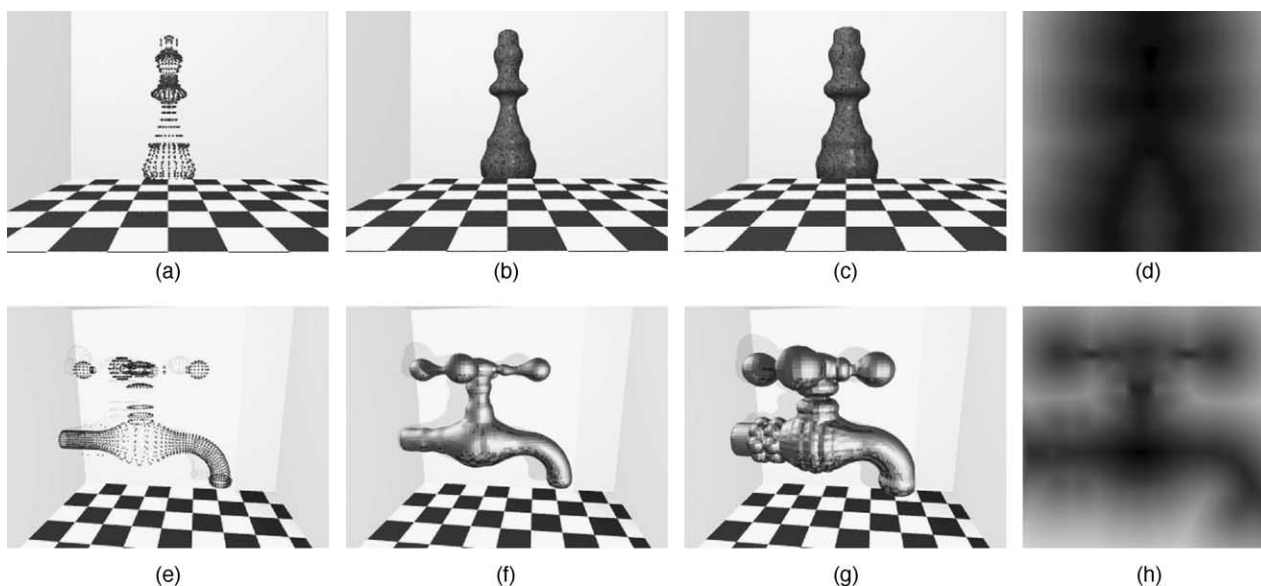


Fig. 8. Examples of shape reconstruction from scattered data points (b) and (c) are iso-surface at different intensity values of the object reconstructed from point set (a); (f) and (g) represent the reconstructed shape from dataset (e) at different iso-values; (d) and (h) are cross-section views.

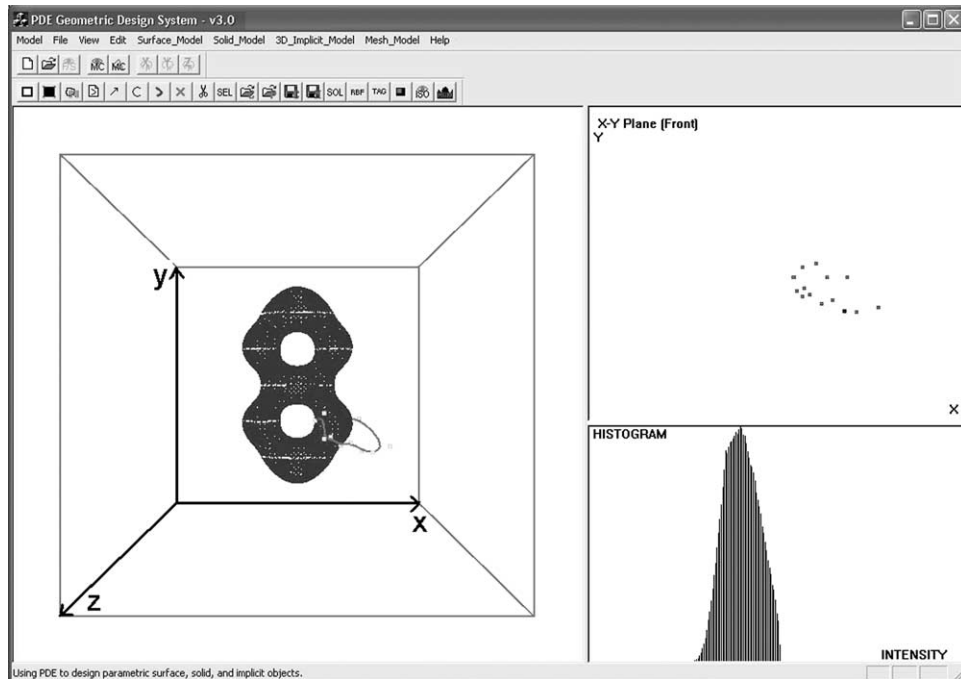


Fig. 9. Snapshot of the interface of our implicit PDE system.

defines the inside and outside of the object. Without the predefinition of gradient directions, the solution will be trivial. Therefore, gradient information of sketch curves is required for reconstructing a unique shape. Accordingly, changing the gradient directions at selected sketch curves means modifying directions of intensity changes in the implicit working space and will result in different implicit shapes. Our system allows users to specify the gradient direction of each individual sketch curve to construct different implicit PDE objects. Refer to Fig. 10 for examples of specifying and modifying gradient directions of the sketch curves. Without further specification, other examples in this paper have gradient directions pointing inward the curves by default.

### 5.3. Local manipulation of implicit PDE solids

Usually the sketch curve sculpting will deform the entire reconstructed shape, which only offers global

manipulation and is less intuitive for ordinary users to handle. Even with the specification of local areas of interests containing the sculpted sketch curve, the sculpting will affect all the points in the selected regions. Moreover, sometimes the input constraints alone cannot guarantee a satisfactory solution of constructed shape. Therefore, direct modification in selected areas is desirable, especially when the overall recovered shape is satisfactory but minor changes in small localized areas are needed. Our system provides interactive tools for the intensity value modification in selected regions to sculpt the reconstructed shape. The modification will be enforced into Eqs. (6) or (7). Using the aforementioned techniques, we can solve Eqs. (6) and (7) to obtain the modified objects. Because for local manipulations, we only calculate the intensity updates in the neighborhood of selected regions, where the intensity values are governed through the PDE and the selected regions usually have relatively small number of grids comparing

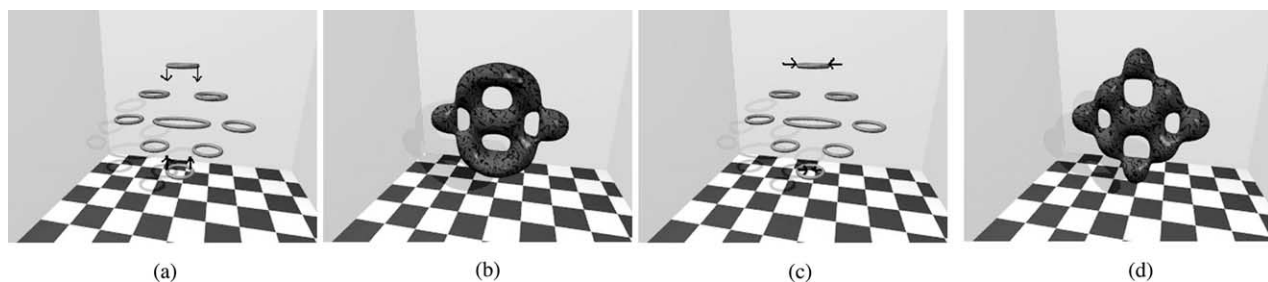


Fig. 10. Examples for specifying and changing gradient directions of sketch curves. (a) and (c) are two sets of curves with same geometric shape but different gradient directions, where the arrows show intensity increasing directions; (b) and (d) are corresponding implicit objects.



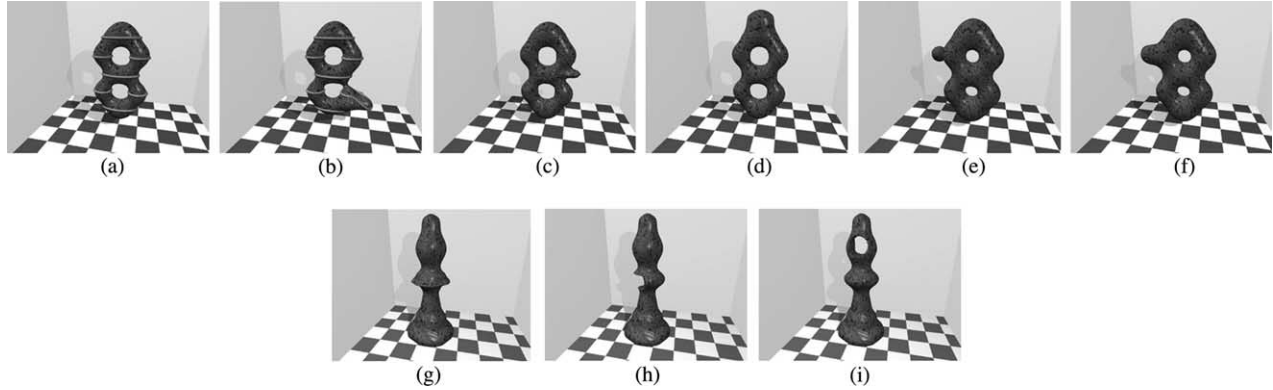


Fig. 11. Examples of enforcing curve and direct manipulation constraints. (a) Original object with sketch curves; (b) deformed object by sculpting a selected curve; (c) changing an iso-contour; (d) deformed object subject to local region constraints; (e) adding a sphere in the working space; and (f) is the corresponding deformed object subject to (e); (g) adding constraints for an object with sharp edges; (h) and (i) are two trimming examples.

with the entire working space, the update of the new intensity values for such regions will be quickly obtained through our finite-difference solver. Therefore, we can achieve interactive manipulations for local sculpting of implicit PDE objects.

Traditional implicit techniques for data reconstruction do not support direct manipulations on the arbitrary locations in the volumetric working space. The changes on the predefined constraints will cause global deformation. It is more desirable to offer users editing functionalities on the interior properties with interactive interface.

### 5.3.1. Local intensity modification

Besides the local RBF approximation for local sketch curve sculpting, our system also allows users to specify any interior region of the sampling grids, and applies intensity changes only within the specified region. Alternatively, we can freeze the selected region and disallow any changes in the specified region. In our system, this can be done through interactively specifying the maximum and minimum sampling grid in  $x$ ,  $y$ , and  $z$  direction of the desired region in the sampling volumetric working space. Subsequently, any change within the region will have no influence on sampling points outside the region. The localized deformation can be easily achieved because only those equations corresponding to the points of the specified regions in Eq. (6) will be solved. In addition, the number of computations is reduced due to fewer number of equations involved in the local sculpting. In principle, all hard constraints can be viewed as some sort of local deformation. Fig. 11 shows examples of local deformation.

### 5.3.2. Iso-surface sculpting

Users can also specify an iso-surface at a particular intensity value and use a cutting plane inside the volumetric working space to get a 2D iso-contour on the plane, then stretch, push, rotate the contour, as well as add desired intensity values at specified locations to modify the shape of

the iso-surface and the intensity distribution of the interested areas. Refer to Fig. 11 for illustrative examples.

### 5.3.3. CSG operations

We also offer several CSG sculpting tools such as using spheres and cubes to trim/extrude/sculpt implicit objects by adding more constraints on the sampling grids of the working space. This is extremely useful for such situations when there are some minor changes needed to be done in some local small regions. Such sculpting tools make our system compatible with CSG-based implicit models by treating those models as modeling tools. Examples are shown in Fig. 11.

### 5.3.4. Gradient constraints

The intensity gradient  $\nabla$  at a point  $(x, y, z)$  in the intensity field can be defined as

$$\nabla d(x, y, z) = \left( \frac{\partial d(x, y, z)}{\partial x}, \frac{\partial d(x, y, z)}{\partial y}, \frac{\partial d(x, y, z)}{\partial z} \right).$$

By applying the finite-difference techniques, the gradient vector  $\nabla d_{i,j,k}$  at a discretized grid point  $\{i, j, k\}$  can be approximated as:

$$\left( \frac{d_{i+1,j,k} - d_{i-1,j,k}}{2\Delta x}, \frac{d_{i,j+1,k} - d_{i,j-1,k}}{2\Delta y}, \frac{d_{i,j,k+1} - d_{i,j,k-1}}{2\Delta z} \right).$$

It provides information about intensity changes in the neighborhood of  $(x, y, z)$  in the working space. Therefore, changing the direction and length of the gradient vector of a selected grid point will affect the intensity distribution in its neighborhood, and as a result, deform the object. Our system allows users to pick a point inside the working space, specify the local region surrounding the point, and modify its gradient vector interactively, then the shape bounded by the specified local region will be deformed accordingly (refer to Figs. 12 and 13(a)).

### 5.3.5. Curvature constraints

The mean curvature at point  $(x, y, z)$  in the intensity field can be computed from the divergence of the intensity

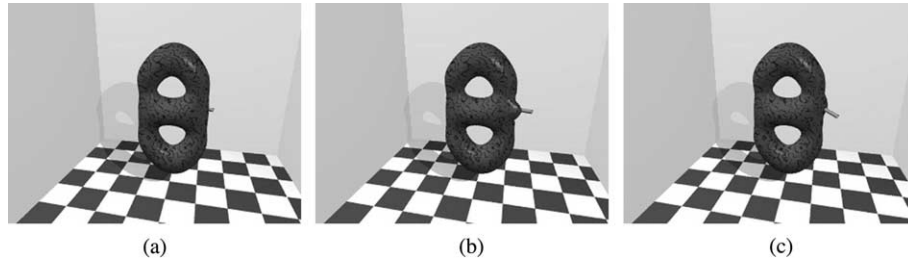


Fig. 12. Examples of enforcing gradient constraints. (a) Original object with the gradient vector at a selected point; (b) and (c) are deformed objects by changing the gradient at the point.

gradient of  $(x, y, z)$ , i.e.  $\nabla \cdot \nabla d(x, y, z)$  [33]. In the discretized form, it can be approximated as  $\nabla \cdot \nabla d_{i,j,k}$ . Its definition is also related to the intensity value of the point's neighbors. By changing the curvature value at a point, the shape of the object will be changed. Our system allows users to manipulate the curvature at a selected grid point for implicit shape deformation. Fig. 13 shows examples.

## 6. Implementation and discussion

We develop a prototype software system that permits users to reconstruct geometric shapes defined by PDE-based implicit functions from a set of sketch curves, scattered data points, or volumetric datasets. Our system also allows interactive manipulation of reconstructed implicit PDE objects with various intensity constraints in the volumetric working space. The interactive sculpting of implicit PDE objects can be obtained via modification of predefined conditions and interior operations. The system is written in Visual C++ and runs on Windows95/98/NT/2000/XP. Fig. 14 illustrates the architecture of our modeling environment for implicit PDE objects. In particular, our system provides the following functionalities:

*Missing information recovery and shape blending.* The underlying implicit PDEs of our system provide a simple yet systematic mechanism to obtain the volumetric information satisfying specified constraints automatically. Such an advantage makes it possible to recover missing information of input datasets with our system. It can also be used to compute connecting parts between different objects in the working space which leads to shape blending.

*Shape reconstruction.* Users can interactively input and edit scattered data points or sketch curves with specified intensity values, then the system uses the RBF method or distance field approximation to calculate intensity values on the sampling grids within the volumetric working space as initial guesses for the iterative solver of the discretized implicit PDE to obtain approximated solutions for implicit PDE objects satisfying these conditions. Our system can model both close and open implicit shapes.

*Discrete models.* Our system supports implicit PDE objects obtained from solving the fourth-order and second-order elliptic PDEs using: (1) finite-difference discretization for the numerical solution of the elliptic PDEs in 3D working space; and (2) RBF approximation at arbitrary sub-regions in the working space for modeling localized details and performance speedup.

*Interactive and direct operations.* Users can also work directly on implicit PDE objects through: (1) local modification of blending coefficient functions; (2) sketch curve sculpting using B-spline manipulation; (3) gradient specification of selected curves; (4) local RBF approximation for improved time performance and interactive CSG manipulation; (5) interior deformation with additional constraints inside the working space; (6) iso-surface manipulation and direct manipulation of iso-contours at selected intensity values; and (7) gradient and curvature constraints inside the working space.

We employ iterative methods (e.g. Gauss-Seidel iteration) with multi-grid-like techniques to solve the implicit PDEs subject to various constraints. Besides original datasets or predefined sketch curves, our system allows users to interactively define and sculpt sketch

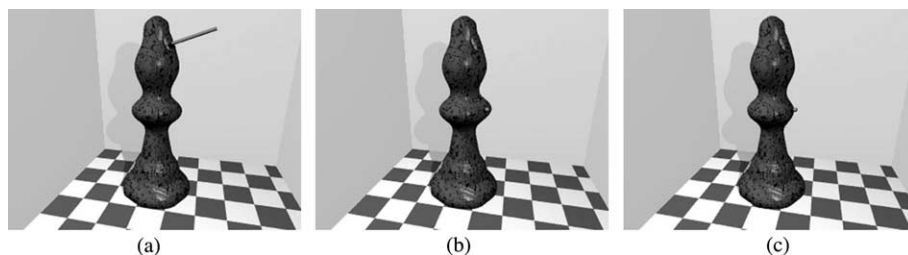


Fig. 13. Examples of enforcing gradient and curvature constraints. (a) Deformed object by changing gradient; (b) and (c) are deformed objects by changing curvature at a selected point (shown in green) from (a).

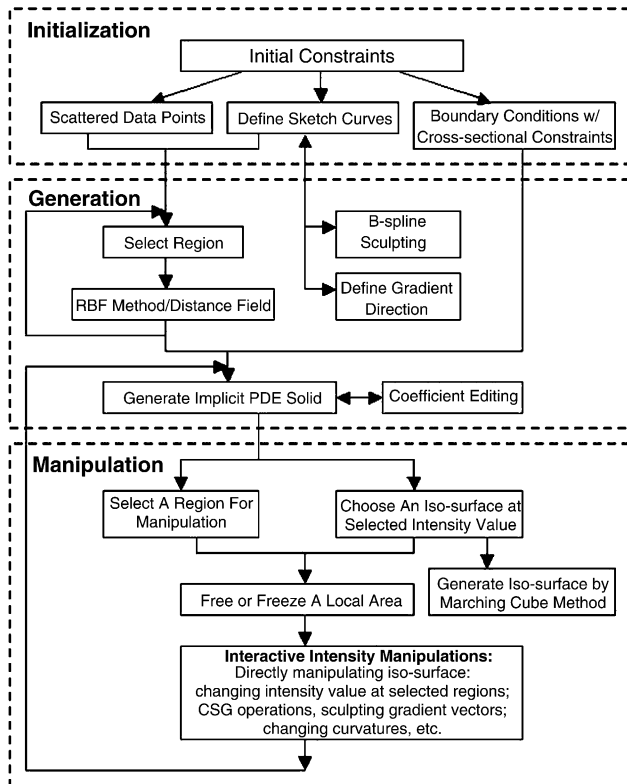


Fig. 14. System architecture and functionalities.

curves directly and specify gradients at selected curves. These constraints provide more freedom to designers and make intuitive design of implicit objects more cost-effective. We also enforce additional constraints directly inside the volumetric working space, apply local operations, and provide sculpting toolkits for the implicit objects, which facilitate the construction of implicit PDE objects of arbitrary topology. The PDE is solved by finite-difference techniques because they are simple, easy to implement, and suitable for complicated, flexible constraints. In general, the time and space complexity are increased with higher resolution as well as increased accuracy. Examples in this paper are rendered by POV-RAY.

Table 1 summarizes the numbers of constraints and CPU time of numerical solvers for the second-order and

fourth-order implicit PDE examples when running on a Pentium 4.1.4 GHz PC. The resolution of the working space is  $64 \times 64 \times 64$  for Fig. 3 and  $65 \times 65 \times 65$  for other examples. The stopping threshold (difference between two iteration steps) is  $10^{-9}$ . 'Initial' stands for the initial guess where we use RBF method for sketch curve datasets and fast-tagging approximation for the scattered data points input. '2nd(s)' and '4th(s)' indicate the CPU time in seconds for solving the entire implicit second-order and fourth-order PDE working spaces based on the initial guess using multi-grid Gauss-Seidel iteration. The time performance of RBF and fast-tagging algorithms depends on the number of enforced constraints, while the convergent speeds of iterative methods are mainly determined by the sampling rates of the implicit working space.

Although the initialization of the implicit models are time-consuming because of the approximation of the entire working space, the local sculpting afterward will be interactive because only small number of sampling grids are involved. Table 2 summarizes the CPU time for the examples of our direct sculpting in local selected regions. 'Cons' stands for the number of constraints involved for the operation, 'Grids' represents the number of grid points in the selected region, and '4th(s)' gives the CPU time (seconds) for updating the intensity values in the selected area using the fourth-order PDE. The CPU time depends on the scale of the intensity change by the sculpting operation as well as the number of constraints and the size of the selected region. For instance, CSG operations usually enforce relatively larger intensity changes for constraints in selected regions than other operations such as gradient and curvature sculpting, hence they need more CPU time to update the region's intensity values.

Despite the direct and powerful modeling advantages of our PDE framework, the major difficulty associated with our PDE techniques is the convergent speed of finite-difference approximation for initial shapes. Thus, faster numerical approximation techniques for solving PDEs need to be considered to improve the time performance of our PDE modeling system.

Table 1  
CPU time (seconds) of different solvers for several examples of implicit PDE objects with different number of constraints

Examples	Constraints	Initial	2nd(s)	4th(s)
Fig. 3	169888	N/A	1.542	7.992
Fig. 4	274086	N/A	3.04	13.7
Fig. 6a	180	5.889	N/A	379.766
Fig. 6b	720	18.872	N/A	416.312
Fig. 8a	1219	267.925	N/A	113.432
Fig. 8c	3154	359.657	N/A	148.283

Table 2  
CPU time (seconds) of local direct manipulation examples of implicit PDE objects

Examples	Cons	Grids	4th(s)
Iso-contour Editing (Fig. 10c)	8	1792	0.45
Region Deformation (Fig. 10d)	507	6358	3.17
CSG-like Blending (Fig. 10f)	108	1000	1.15
Sharp-feature Creation (Fig. 10g)	98	5046	0.23
Cutting-1 (Fig. 10h)	216	1000	0.82
Cutting-2 (Fig. 10i)	216	1000	0.82
Gradient Sculpting (Fig. 12)	7	294	0.09
Curvature Manipulation (Fig. 13)	7	294	0.09

## 7. Conclusion

We have unified the popular implicit function techniques with the powerful parametric PDE framework to demonstrate more modeling advantages of the PDE-based paradigm. Our prototype system supports interactive shape design of implicit PDE objects through global and local deformation of scattered data points or sketch curves. The implicit PDE model can be defined as the solution of the elliptic PDEs over a scalar intensity field with either scattered-point datasets or a set of sketch curves as generalized boundary and additional constraints. Our implicit PDE approach can also provide an approximation for the missing or blending part in the working space with most of the intensity information already known. Our software environment offers users a set of interactive and direct shape modeling toolkits including: sketch curve sculpting and gradient manipulation, intensity value modification in selected regions, gradient and curvature manipulations inside the working space, and iso-contour manipulation of specified intensity value inside the volumetric domain. These toolkits provide users an intuitive interface to model implicit PDE objects satisfying a set of design criteria and functional requirements. Our integrated approach and novel PDE techniques further expand the geometric coverage and the topological flexibility of the conventional PDE methodology to implicit functions, and forge ahead toward the realization of the full potential of PDE technology in shape modeling and other visual computing fields.

## Acknowledgements

This research was supported in part by the NSF ITR grant IIS-0082035, the NSF grant IIS-0097646, the NFS grant CCR-0328930, the NSF ITR grant IIS-0326388, Alfred P. Sloan Fellowship, and Honda Initiation Award.

## References

- [1] Bærentzen A, Christensen N. Volume sculpting using level-set method. In *Shape Modeling International* 2002, Banff, Alberta, Canada; 2002. p. 175–82.
- [2] Bertalmio M, Sapiro G, Caselles V, Ballester C. Image inpainting. In *SIGGRAPH* 2000, New Orleans, USA; 2000. p. 417–24.
- [3] Bloomenthal J, Bajaj C, Blinn J, Cani-Gascuel M-P, Rockwood A, Wyvill B, Wyvill G. *Introduction to Implicit Surfaces*. Los Altos, CA: Morgan Kaufmann; 1997.
- [4] Bloomenthal J, Wyvill B. Interactive techniques for implicit modeling. *Comput Graphics* 1990;24(2):109–16.
- [5] Bloor MIG, Wilson MJ. Generating blend surfaces using partial differential equations. *Comput Aided Des* 1989;21(3):165–71.
- [6] Bloor MIG, Wilson MJ. Using partial differential equations to generate free-form surfaces. *Comput Aided Des* 1990;22(4):202–12.
- [7] Bloor MIG, Wilson MJ. Functionality in solids obtained from partial differential equations. *Computing Suppl* 1993;8:21–42.
- [8] Breen D, Whitaker R. A level-set approach for the metamorphosis of solid models. *IEEE Transact Vis Comput Graphics* 2001;7(2):173–92.
- [9] Carr J, Beatson R, Cherrie J, Mitchell T, Fright W, McCallum B. Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH* 2000, Los Angeles, USA; 2001. p. 67–76.
- [10] Cohen-Or D, Levin D. Three-dimensional distance field metamorphosis. *ACM Transact Graphics* 1998;17(2):116–41.
- [11] Cutler B, Dorsey J, McMillan L, Müller M, Jagnow R. A procedural approach to authoring solid models. In *SIGGRAPH* 2002, San Antonio, TX; 2002. 302–11.
- [12] Desbrun M, Cani-Gascuel M-P. Active implicit surfaces for animation. *Graphics Interface* 1998;143–50.
- [13] Desbrun M, Tsingos N, Gascuel M-P. Adaptive sampling of implicit surfaces for interactive modelling and animation. *Comput Graphics Forum* 1996;15(5):319–25.
- [14] Du H, Qin H. Direct manipulation and interactive sculpting of PDE surfaces. *Comput Graphics Forum* 2000;19(3):C261–70.
- [15] Du H, Qin H. Dynamic PDE surfaces with flexible and general constraints. In *Pacific Graphics* 2001, Hong Kong; 2000. p. 213–22.
- [16] Du H, Qin H. Integrating physics-based modeling with PDE solids for geometric design. In *Pacific graphics*, Tokyo, Japan; 2001. p. 198–207.
- [17] Ebert DS, Musgrave FK, Prusinkiewicz P, Stam J, Tessendorf J. Simulating nature: from theory to practice. *SIGGRAPH* 2000 Course notes 25; 2000.
- [18] Ferley E, Cani M, Gascuel J. Practical volumetric sculpting. *Visual Comput* 2000;16(8):469–80.
- [19] Foster N, Metaxas D. Realistic animation of liquids. In *Proceedings of GI*; 1996. p. 204–12.
- [20] Foster N, Metaxas D. Modeling the motion of hot, turbulent gas. In *SIGGRAPH* 1997, Los Angeles, CA, USA; 1997. p. 181–8.
- [21] Frisken S, Perry R, Rockwood A, Jones T. Adaptive sampled distance fields: a general representation of shape for computer graphics. In *SIGGRAPH* 2000, New Orleans, USA; 2000. p. 249–54.
- [22] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized points. In *SIGGRAPH* 1992; 1992. p. 71–8.
- [23] Hua J, Qin H. Haptic sculpting of volumetric implicit functions. In *Pacific Graphics* 2001, Tokyo, Japan; 2001. p. 254–64.
- [24] Hua J, Qin H. Dynamic implicit solids with constraints for haptic sculpting. In *Shape Modeling International* 2001, Banff, Alberta, Canada; 2002. p. 119–28.
- [25] Lorensen W, Cline H. Marching cubes: a high resolution 3D surface construction algorithm. In *SIGGRAPH* 1987; 1987. p. 163–9.
- [26] Morse B, Yoo T, Rheingans P, Chen D, Subramanian K. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modeling International* 2001, Genova, Italy; 2001. p. 89–98.
- [27] Muraki S. Volumetric shape description of range data using blobby model. *Comput Graphics* 1991;25(4):227–35.
- [28] Museth K, Breen D, Whitaker R, Barr A. Level set surface editing operators. In *SIGGRAPH* 2002, San Antonio, TX, USA; 2002. p. 330–8.
- [29] Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel H-S. Multi-level partition of unity implicits. In *SIGGRAPH* 2003, San Diego, USA; 2003. p. 463–70.
- [30] Perry R, Frisken S. Kizamu: a system for sculpting digital characters. In *SIGGRAPH* 2001, Los Angeles, USA; 2001. p. 47–56.
- [31] Press WH, Teulolsky SA, Vetterling WT, Flannery BP. *Numerical recipes in C*. Cambridge University Press; 1993.
- [32] Raviv A, Elber G. Three dimensional freeform sculpting via zero sets of scalar trivariate functions. In *Proceedings of 5th ACM Symposium on Solid Modeling and Applications*, Ann Arbor, Michigan, United States; 1999. p. 246–257.



- [33] Sarti A, Tubaro S. Multiresolution implicit object modeling. In VMV 2001, Stuttgart, Germany; 2001. p. 93–100.
- [34] Savchenko VV, Pasko AA, Okunev OG, Kunii TL. Function representation of solids reconstructed from scattered surface points and contours. *Comput Graphics Forum* 1995;14(4):181–8.
- [35] Schneider R, Kobbelt L. Generating fair meshes with G1 boundary conditions. In *Geometric Modeling and Processing Conference Proceedings*; 2000. p. 251–61.
- [36] Sclaroff S, Pentland A. Generalized implicit functions for computer graphics. *Comput Graphics* 1991;25(4):247–50.
- [37] Stam J. Stable fluids. In *SIGGRAPH 1999*, Los Angeles, CA, USA; 1999. p. 121–7.
- [38] Strang G. *Introduction to applied mathematics*. Wellesley-Cambridge Press; 1986.
- [39] Turk G, Dinh HQ, O'Brien J. Implicit surfaces that interpolate. In *Shape Modeling International 2001*, Genova, Italy; 2001. p. 62–71.
- [40] Turk G, O'Brien J. Shape transformation using variational implicit functions. In *SIGGRAPH 1999*, Los Angeles, CA, USA; 1999. p. 335–42.
- [41] Turk G, O'Brien J. Modeling with implicit surfaces that interpolate. *ACM Transact Graphics* 2002;21(4):855–73.
- [42] Ugail H, Bloor MIG, Wilson MJ. Techniques for interactive design using the PDE method. *ACM Transact Graphics* 1999;18(2):195–212.
- [43] Whitaker R, Breen D. Level-set models for the deformation of solid objects. In *Conference of Implicit Surface 1998*, Seattle, USA; 1998. p. 19–36.
- [44] Witkin A, Heckbert P. Using particles to sample and control implicit surfaces. In *SIGGRAPH 1994*; 1994. p. 269–77.
- [45] Zhang JJ, You L. Surface representation using second, fourth and mixed order partial differential equations. In *Shape Modeling International 2001*, Genova, Italy; 2001. p. 250–6.
- [46] Zhao HK, Osher S, Fedkiw R. Fast surface reconstruction using level set method. In *IEEE Workshop on Variational and Level Set Methods (VLSM 01)* Vancouver, Canada; 2001. p. 194–202.
- [47] Zhao HK, Osher S, Merriman B, Kang M. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Comput Vision Image Understanding* 2000;80(3):295–319.



**Haixia Du** is a PhD candidate in the Computer Science Department at the State University of New York (SUNY) at Stony Brook, where she is also a Research Assistant in the Center for Visual Computing (CVC), SUNY at Stony Brook. Haixia received her BS degree in Computer Science from Jilin University in Changchun, P.R. China in 1995 and her ME degree in Computer Science from Institute of Mathematics, Chinese Academy of Sciences in Beijing, P.R. China in 1998. She also received her MS degree in Computer Science from SUNY at Stony Brook in 2000. Her research interests include geometric and physics-based modeling, computer animation and simulation, visualization, and computer graphics. For more information, see <http://www.cs.sunysb.edu/~dhaixia>



**Dr Hong Qin** is an Associate Professor of Computer Science at the State University of New York at Stony Brook, where he is also a member of the SUNYSB Center for Visual Computing. He received his BS degree (1986) and his MS degree (1989) in Computer Science from Peking University in Beijing, P.R. China. He received his PhD degree (1995) in Computer Science from the University of Toronto. From 1989–1990 he was a research scientist at North-China Institute of Computing Technologies. From 1990–1991 he was a PhD candidate in Computer Science at the University of North Carolina at Chapel Hill. From 1996–1997, he was an Assistant Professor of Computer and Information Science and Engineering at the University of Florida. In 1997, Dr Qin was awarded the NSF CAREER Award from the National Science Foundation (NSF), and, in September, 2000, was awarded a newly-established NSF Information Technology Research (ITR) grant. In December, 2000, he received a Honda Initiation Grant Award, and, in April, 2001, was selected as an Alfred P. Sloan Research Fellow by the Sloan Foundation. He is a member of ACM, IEEE, SIAM, and Eurographics.



# SIGGRAPH2005

## Motivation



SIGGRAPH2005

- Desirable property for an implicitly defined surface:  
**Locality**
  - Trade-off between accuracy and smoothing  
Why? Noisy input data
  - Efficient processing of input points  
Why? Large number of points
  - Efficient computation of scalar values  
Why? Large number of evaluations



## Motivation



- Desirable property for an implicitly defined surface:  
**Locality**
  - Trade-off between accuracy and smoothing  
Why? Noisy input data
  - Efficient processing of input points  
Why? Large number of points
  - Efficient computation of scalar values  
Why? Large number of evaluations



## Introduction & Basics



- Introduction & Basics
  - Notation, functional approximation
- Multi-level Partition of Unity Implicits
  - Implicit approximation, sharp features, computation, results
- Extensions
  - Other spatial arrangements
  - Faster rendering

## Introduction & Basics



- Notation, Terms
  - Regular/Irregular, Approximation/Interpolation, Global/Local
- Standard interpolation/approximation techniques
  - Global: Triangulation, Voronoi-Interpolation, Least Squares (LS), Radial Basis Functions (RBF)
  - Local: Shepard/Partition of Unity Methods, Moving LS
- Problems
  - Sharp edges, feature size/noise
- Functional -> Manifold

## Introduction & Basics

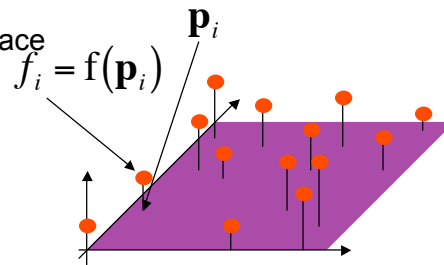


- Notation, Terms
  - Regular/Irregular, Approximation/Interpolation, Global/Local
- Standard interpolation/approximation techniques
  - Global: Triangulation, Voronoi-Interpolation, Least Squares (LS), Radial Basis Functions (RBF)
  - Local: Shepard/Partition of Unity Methods, Moving LS
- Problems
  - Sharp edges, feature size/noise
- Functional -> Manifold

## Notation



- Consider functional (height) data for now
- Data points are represented as
  - Location in parameter space
  - With certain height

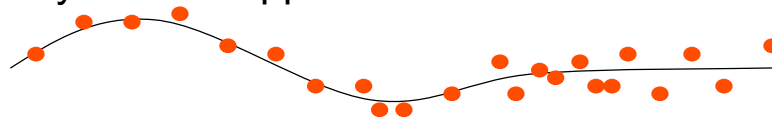


- Goal is to approximate  $f$  from  $f_i, p_i$

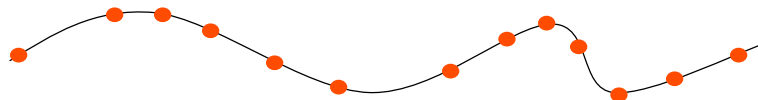
## Terms: Approximation/Interpolation



- Noisy data  $\Rightarrow$  Approximation



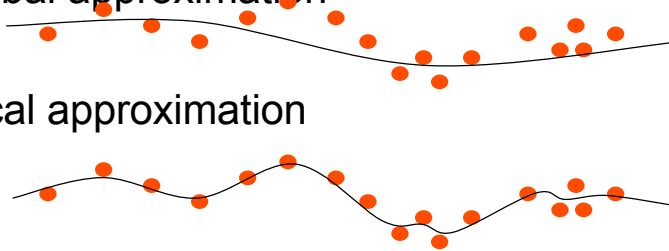
- Perfect data  $\Rightarrow$  Interpolation



## Terms: Global/Local



- Global approximation
- Local approximation
- Locality comes at the expense of fairness



## Introduction & Basics

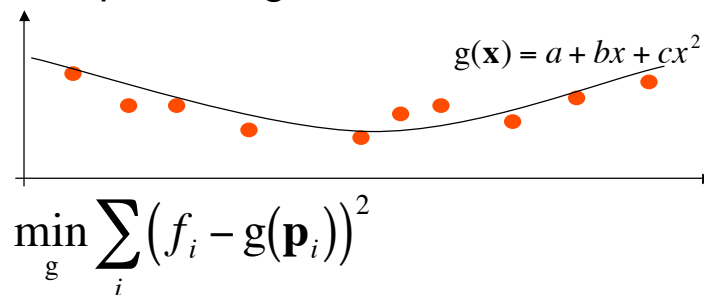


- Terms
  - Regular/Irregular, Approximation/Interpolation, Global/Local
- Standard interpolation/approximation techniques
  - Global: Triangulation, Voronoi-Interpolation, Least Squares (LS), Radial Basis Functions (RBF)
  - Local: Shepard/Partition of Unity Methods, Moving LS
- Problems
  - Sharp edges, feature size/noise
- Functional -> Manifold

## Least Squares



- Fits a primitive to the data
- Minimizes squared distances between the  $\mathbf{p}_i$ 's and primitive  $g$



## Least Squares - Example



- Primitive is a (univariate) polynomial

$$g(x) = (1, x, x^2, \dots) \cdot \mathbf{c}^T$$

- $$\min \sum_i \left( f_i - (1, p_i, p_i^2, \dots) \mathbf{c}^T \right)^2 \Rightarrow$$

$$0 = \sum_i 2 p_i^j \left( f_i - (1, p_i, p_i^2, \dots) \mathbf{c}^T \right)$$

- Linear system of equations

## Least Squares - Example



SIGGRAPH2005

- Resulting system

$$0 = \sum_i 2p_i^j \left( f_i - (1, p_i, p_i^2, \dots) \mathbf{c}^T \right) \Leftrightarrow$$

$$\sum_i \begin{pmatrix} 1 & p_i & p_i^2 & \dots \\ p_i & p_i^2 & p_i^3 & \\ p_i^2 & p_i^3 & p_i^4 & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \end{pmatrix} = 2 \sum_i f_i \begin{pmatrix} 1 \\ p_i \\ p_i^2 \\ \vdots \end{pmatrix}$$

## Radial Basis Functions



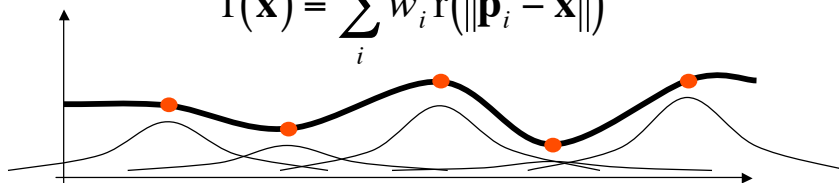
SIGGRAPH2005

- Represent approximating function as

- Sum of radial functions  $r$

- Centered at the data points  $p_i$

$$f(\mathbf{x}) = \sum_i w_i r(\|\mathbf{p}_i - \mathbf{x}\|)$$





## Radial Basis Functions



- Solve  $f_j = \sum_i w_i r(\|\mathbf{p}_i - \mathbf{p}_j\|)$

to compute weights  $w_i$

- Linear system of equations

$$\begin{pmatrix} r(0) & r(\|\mathbf{p}_0 - \mathbf{p}_1\|) & r(\|\mathbf{p}_0 - \mathbf{p}_2\|) & \cdots \\ r(\|\mathbf{p}_1 - \mathbf{p}_0\|) & r(0) & r(\|\mathbf{p}_1 - \mathbf{p}_2\|) & \\ r(\|\mathbf{p}_2 - \mathbf{p}_0\|) & r(\|\mathbf{p}_2 - \mathbf{p}_1\|) & r(0) & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \end{pmatrix}$$

## Radial Basis Functions



- Solvability depends on radial function
- Several choices assure solvability
  - $r(d) = d^2 \log d$  (thin plate spline)
  - $r(d) = e^{-d^2/h^2}$  (Gaussian)
    - $h$  is a data parameter
    - $h$  reflects the feature size or anticipated spacing among points



SIGGRAPH2005

## Function Spaces!

---

- Monomial, Lagrange, RBF share the same principle:
  - Choose basis of a function space
  - Find weight vector for base elements by solving linear system defined by data points
  - Compute values as linear combinations
- Properties
  - One costly preprocessing step
  - Simple evaluation of function in any point



SIGGRAPH2005

## Function Spaces?

---

- Problems
  - Many points lead to large linear systems
  - Evaluation requires global solutions
- Solutions
  - RBF with compact support
    - Matrix is sparse
    - Still: solution depends on every data point, though drop-off is exponential with distance
  - Local approximation approaches

## Introduction & Basics

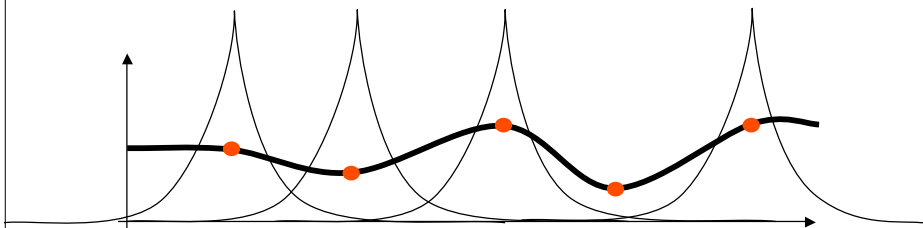


- Terms
  - Regular/Irregular, Approximation/Interpolation, Global/Local
- Standard interpolation/approximation techniques
  - Global: Triangulation, Voronoi-Interpolation, Least Squares (LS), Radial Basis Functions (RBF)
  - Local: Shepard/Partition of Unity Methods, Moving LS
- Problems
  - Sharp edges, feature size/noise
- Functional -> Manifold

## Shepard Interpolation



- Approach:  $f(\mathbf{x}) = \sum_i \phi_i(\mathbf{x}) f_i$   
 with basis functions  $\phi_i(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{x}_i\|^{-p}}{\sum_j \|\mathbf{x} - \mathbf{x}_j\|^{-p}}$
- define  $f(\mathbf{p}_i) = f_i = \lim_{\mathbf{x} \rightarrow \mathbf{p}_i} f(\mathbf{x})$



## Shepard Interpolation



- $f(\mathbf{x})$  is a convex combination of  $\phi_i$ ,  
because all  $\phi_i \in [0,1]$  and  $\sum \phi_i(\mathbf{x}) = 1$
- $f(\mathbf{x})$  is contained in the convex hull of data points
- $|\{\mathbf{p}_i\}| > 1 \Rightarrow f(\mathbf{x}) \in C^\infty$  and  $\nabla f(\mathbf{p}_i) = \mathbf{0}$   
→ Data points are saddles
- global interpolation  
→ every  $f(\mathbf{x})$  depends on all data points
- Only constant precision, i.e. only constant functions are reproduced exactly

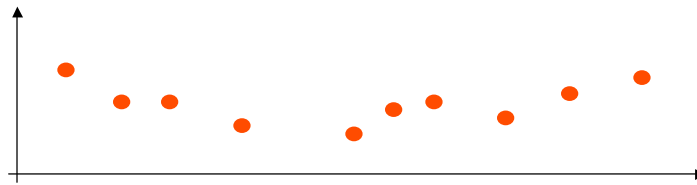
## Shepard Interpolation



Localization:

- Set  $f(\mathbf{x}) = \sum_i \mu_i(\mathbf{x}) \phi_i(\mathbf{x}) f_i$
- with  $\mu_i(\mathbf{x}) = \begin{cases} (1 - \|\mathbf{x} - \mathbf{p}_i\|/R_i)^\nu & \text{if } \|\mathbf{x} - \mathbf{p}_i\| < R_i \\ 0 & \text{else} \end{cases}$   
for reasonable  $R_i$  and  $\nu > 1$   
→ no constant precision because of possible holes in the data

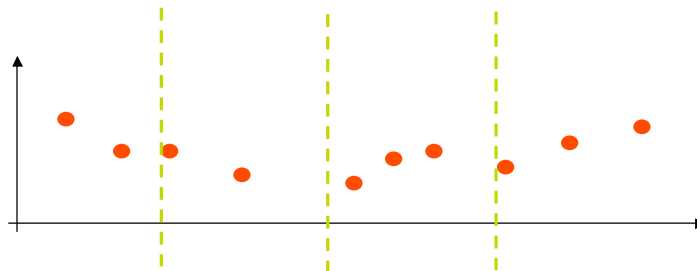
## Partition of Unity Methods



## Partition of Unity Methods



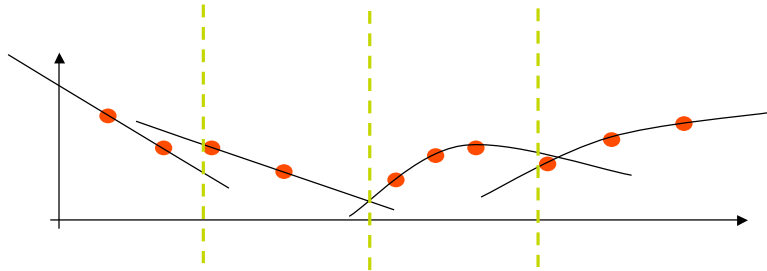
- Subdivide domain into cells



## Partition of Unity Methods



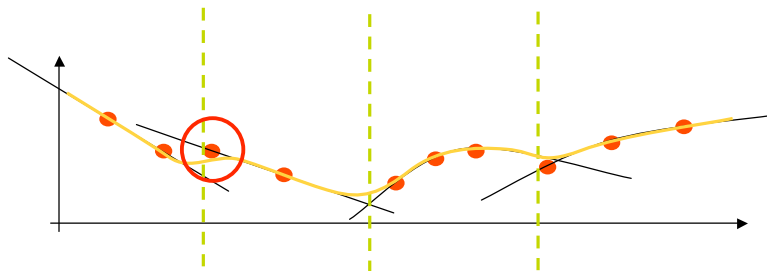
- Compute local interpolation per cell



## Partition of Unity Methods



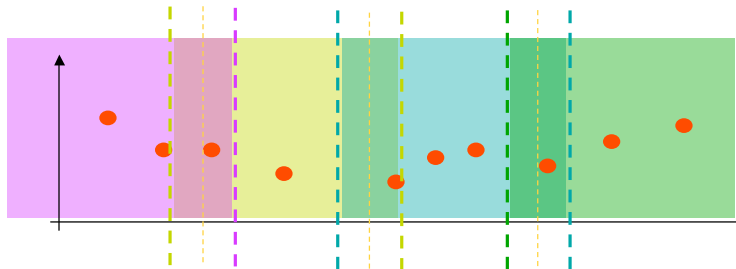
- Blend local interpolations?



## Partition of Unity Methods



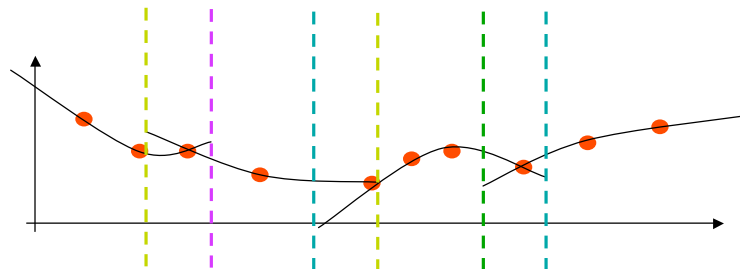
- Subdivide domain into *overlapping* cells



## Partition of Unity Methods



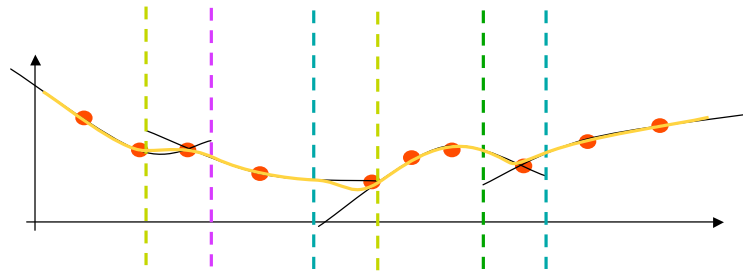
- Compute local interpolations



## Partition of Unity Methods



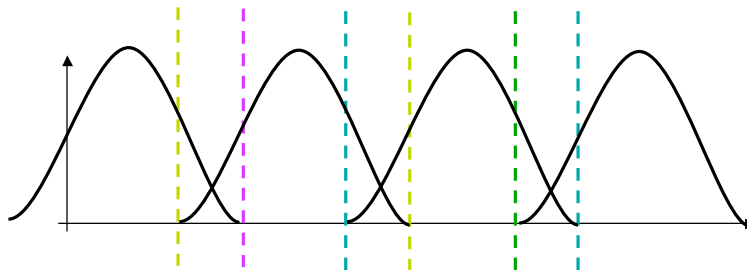
- Blend local interpolations



## Partition of Unity Methods



- Weights should
  - have the (local) support of the cell

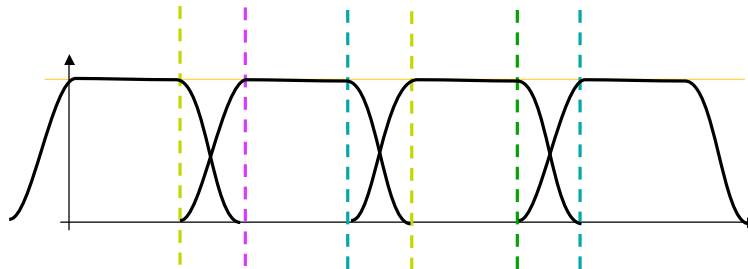




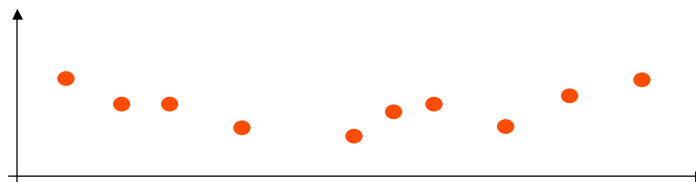
## Partition of Unity Methods



- Weights should
  - sum up to one everywhere (Shepard weights)
  - have the (local) support of the cell



## Hierarchical Approximation

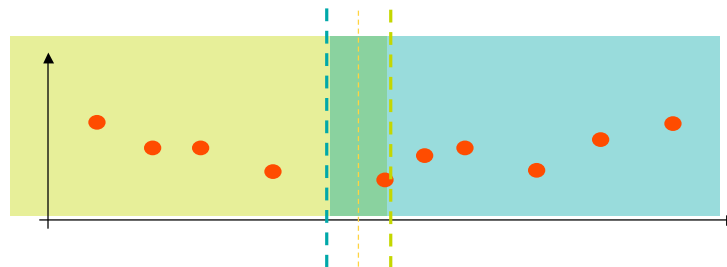


## Hierarchical Approximation



SIGGRAPH2005

- Subdivide domain into overlapping cells

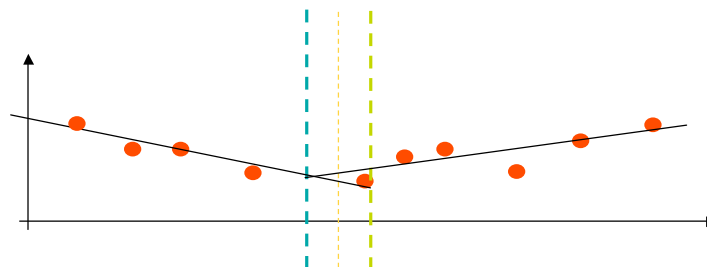


## Hierarchical Approximation



SIGGRAPH2005

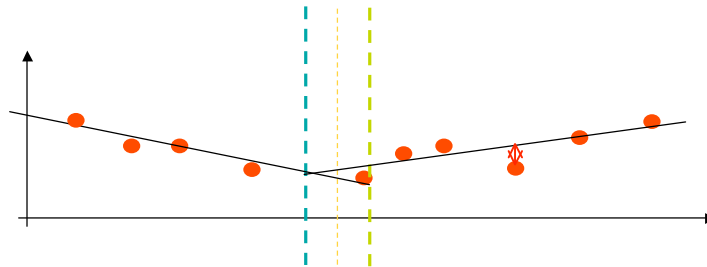
- Compute local least squares approximations



## Hierarchical Approximation



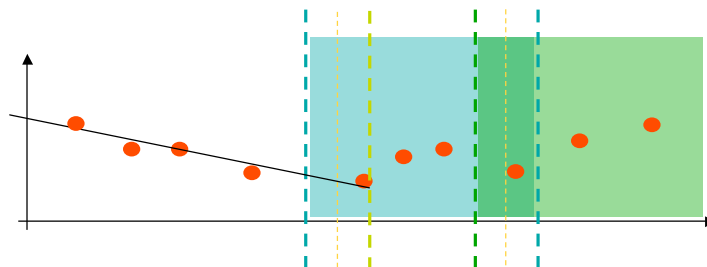
- Compute local approximation error



## Hierarchical Approximation



- Subdivide cells with large error

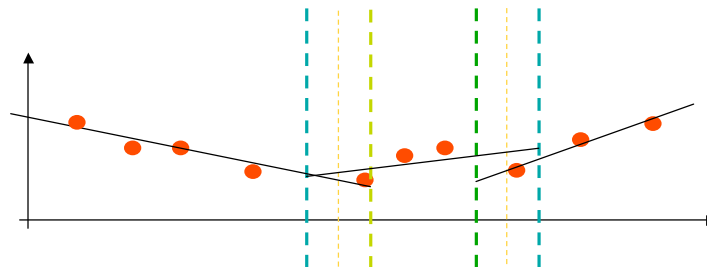


## Hierarchical Approximation



SIGGRAPH2005

- Recompute local approximations per cell

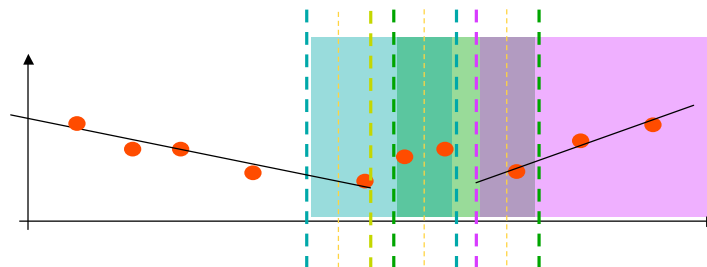


## Hierarchical Approximation



SIGGRAPH2005

- Subdivide again where error is high

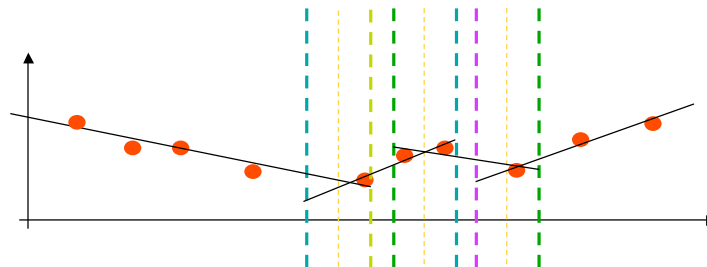


## Hierarchical Approximation



SIGGRAPH2005

- Approximate again

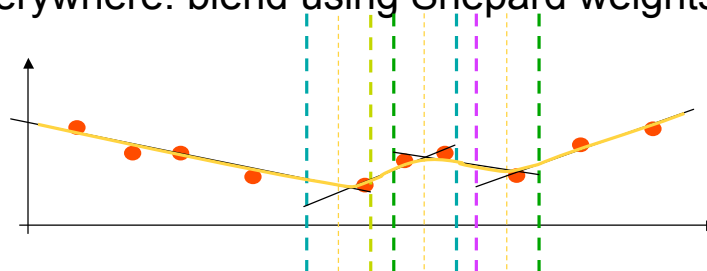


## Hierarchical Approximation



SIGGRAPH2005

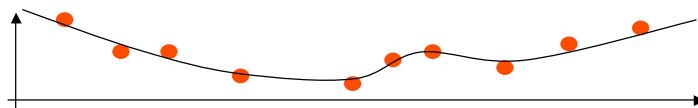
- Once approximation error is bounded everywhere: blend using Shepard weights



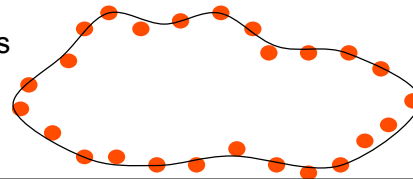
## Functional → Manifold



- Standard techniques are applicable if data represents a function



- Manifolds are more general
  - No parameter domain
  - No knowledge about neighbors



## Overview

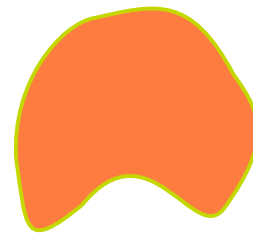


- Introduction & Basics
  - Notation, functional approximation
- Multi-level Partition of Unity Implicits
  - Implicit approximation, sharp features, computation, results
- Extensions
  - Other spatial arrangements
  - Faster rendering

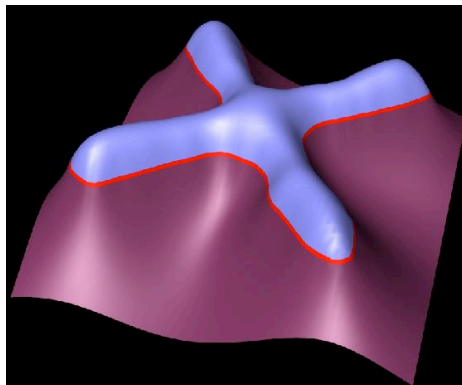
## Implicits



- Idea: Represent 2-manifold as zero-set of a scalar function in 3-space
  - Inside:  $f(\mathbf{x}) < 0$
  - On the manifold:  $f(\mathbf{x}) = 0$
  - Outside:  $f(\mathbf{x}) > 0$



## Implicits - Illustration



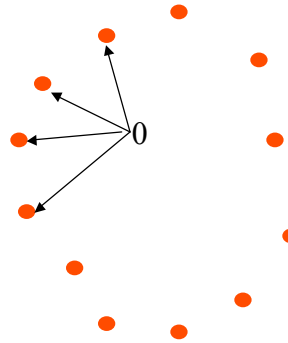
- Image courtesy Greg Turk

## Implicits from point samples



SIGGRAPH2005

- Function should be zero in data points
  - $f(\mathbf{p}_i) = 0$
- Use standard approximation techniques to find  $f$
- Trivial solution:  $f = 0$
- Additional constraints are needed

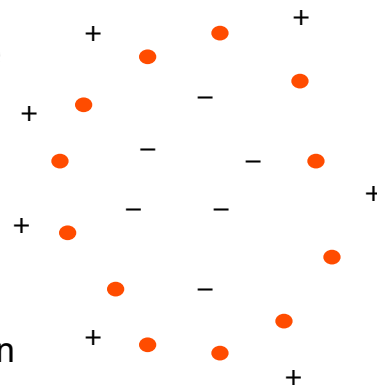


## Implicits from point samples



SIGGRAPH2005

- Constraints define inside and outside
- Simple approach (Turk, O'Brien)
  - Sprinkle additional information manually
  - Make additional information soft constraints



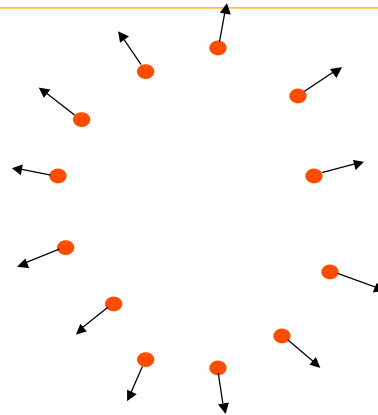


## **Implicits from point samples**



SIGGRAPH2005

- Use normal information
- Normals could be computed from scan
- Or, normals have to be estimated

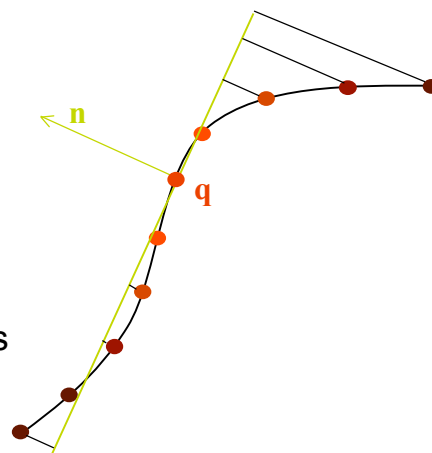


## **Estimating normals**



SIGGRAPH2005

- Normal orientation (Implicits are signed)
  - Use inside/outside information from scan
- Normal direction by fitting a tangent
  - LS fit to nearest neighbors
  - Weighted LS fit
  - MLS fit



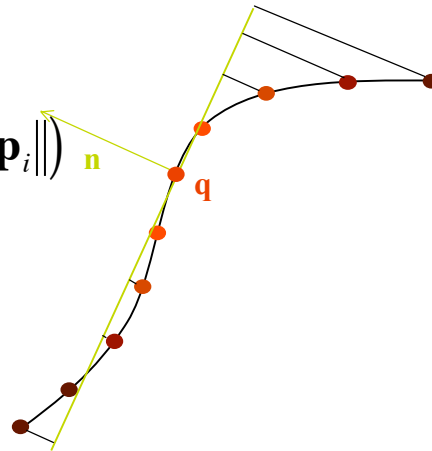
## Estimating normals



- General fitting problem

$$\min_{\|\mathbf{n}\|=1} \sum_i \langle \mathbf{q} - \mathbf{p}_i, \mathbf{n} \rangle^2 \theta(\|\mathbf{q} - \mathbf{p}_i\|)$$

- Problem is non-linear because  $\mathbf{n}$  is constrained to unit sphere



## Estimating normals



- The constrained minimization problem

$$\min_{\|\mathbf{n}\|=1} \sum_i \langle \mathbf{q} - \mathbf{p}_i, \mathbf{n} \rangle^2 \theta_i$$

is solved by the eigenvector corresponding to the smallest eigenvalue of the following co-variance matrix

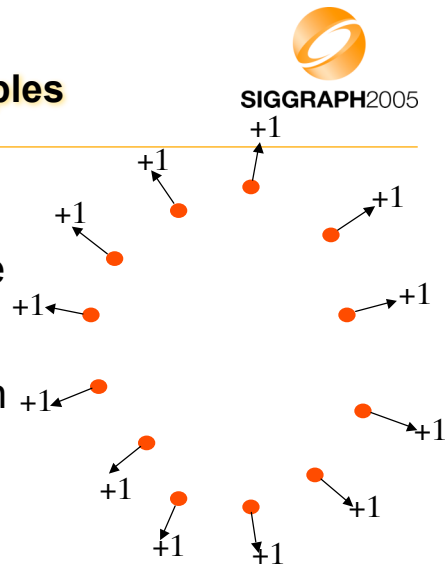
$$\sum_i (\mathbf{q} - \mathbf{p}_i) \cdot (\mathbf{q} - \mathbf{p}_i)^T \theta_i$$

which is constructed as a sum of weighted outer products.

## Implicits from point samples

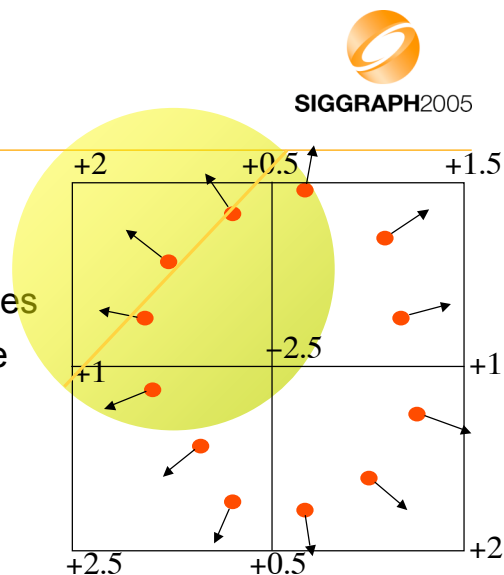
- Compute non-zero anchors in the distance field
- Use normal information directly as constraints

$$f(\mathbf{p}_i + \mathbf{n}_i) = 1$$



## PuO Implicits

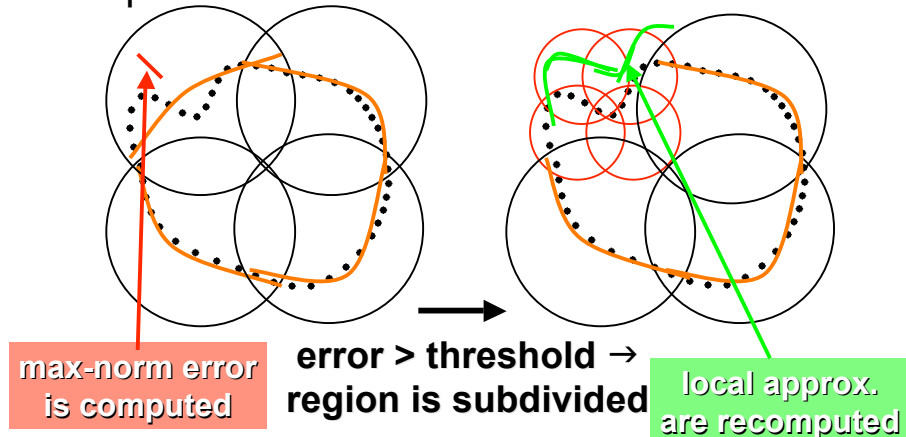
- Construct a spatial subdivision
- Compute distance values
- Compute local distance field approximations
  - e.g. Quadrics
- Blend them with local Shepard weights



## Multi-level PuO Implicits



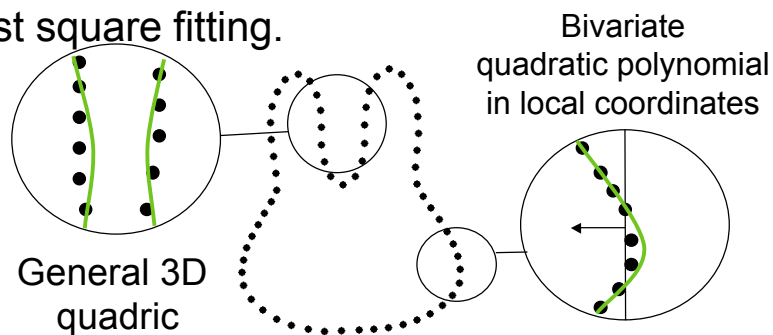
- Adaptive octree subdivision



## Local Approximations



- Second-order polynomial approximations by least square fitting.



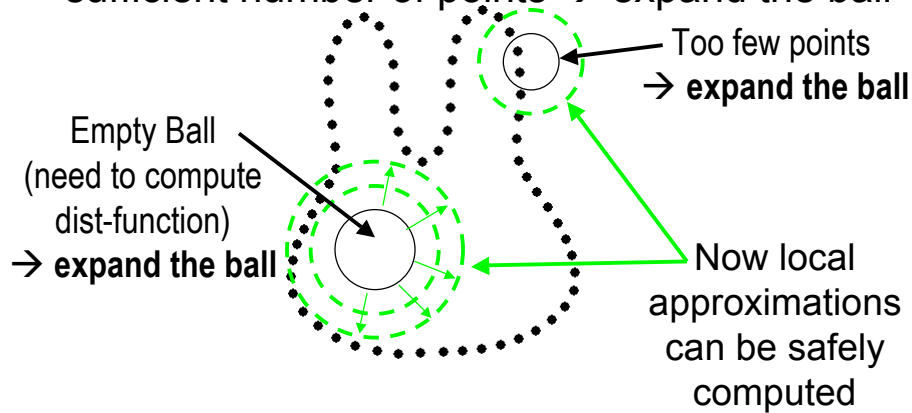
- Approximation type is chosen according to the deviation of normals.

## Local Approximations



SIGGRAPH2005

- If an approximation ball does not contain a sufficient number of points → expand the ball

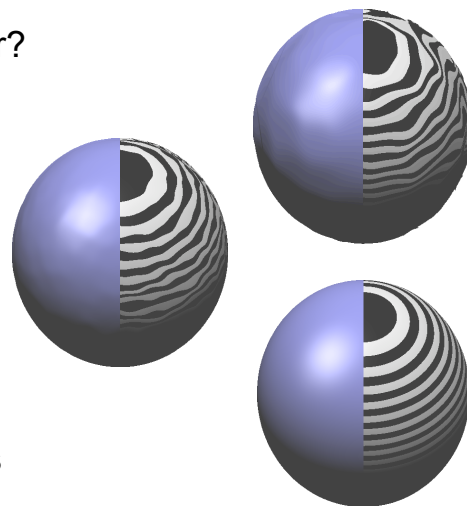


## Local approximations

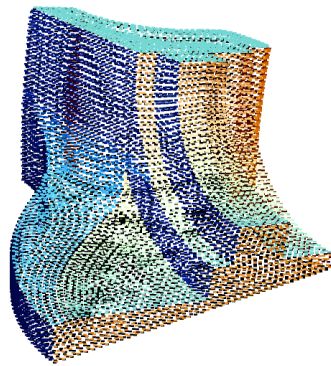


SIGGRAPH2005

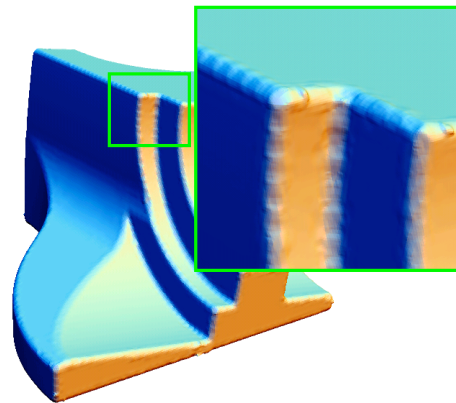
- Degree? Why not linear?
- 128 linear elements
- 450 linear elements
- 105 quadratic elements



## Quadrics & Sharp Features



Points



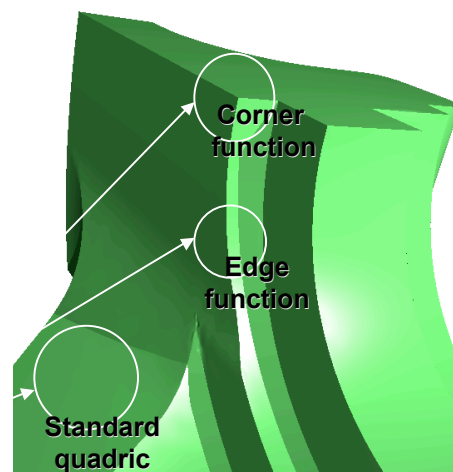
Polygonization of  $f=0$

## Sharp features



Local analysis of  
points and normals

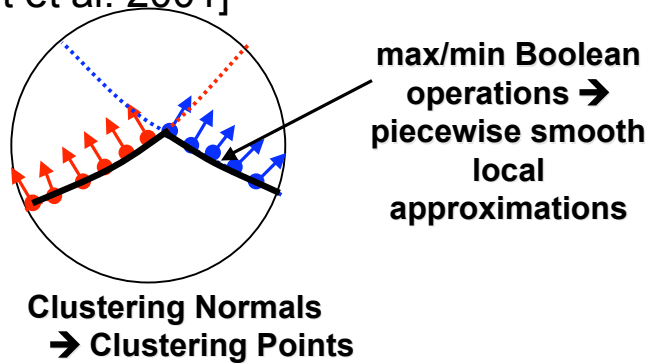
Piecewise quadric  
functions



## Sharp Features



- Edges and corners are recognized by analyzing angles between normals.  
[Kobbelt et al. 2001]

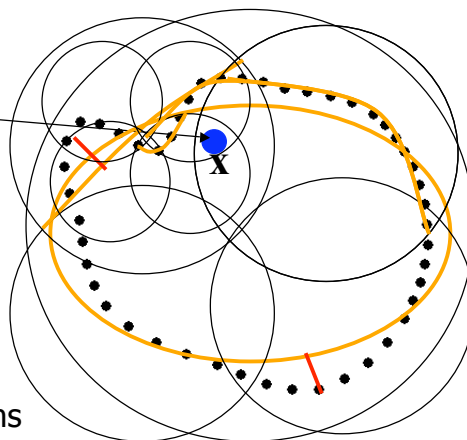


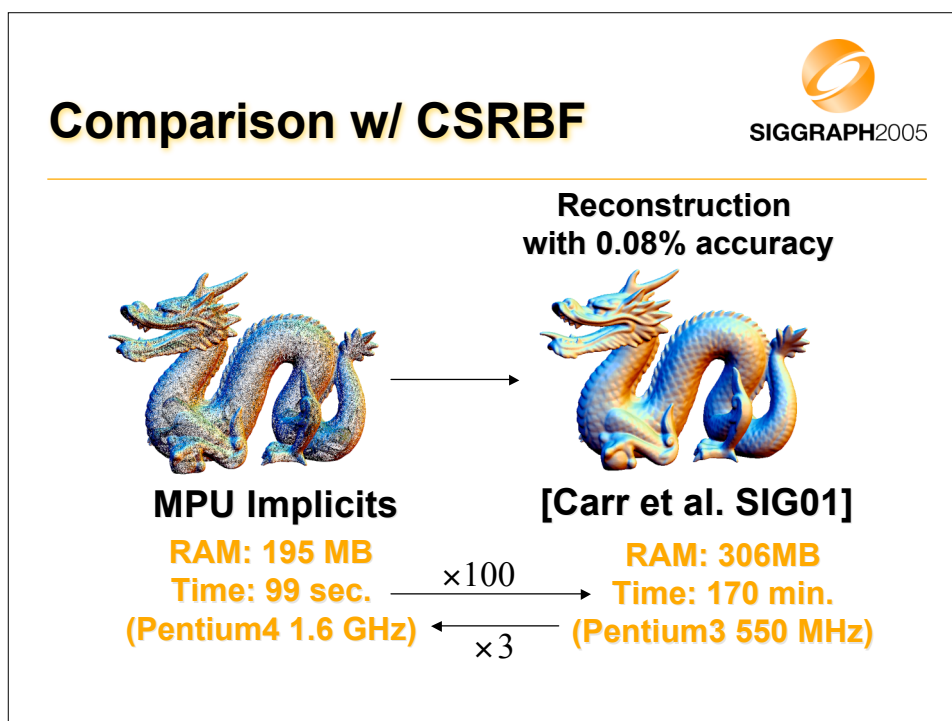
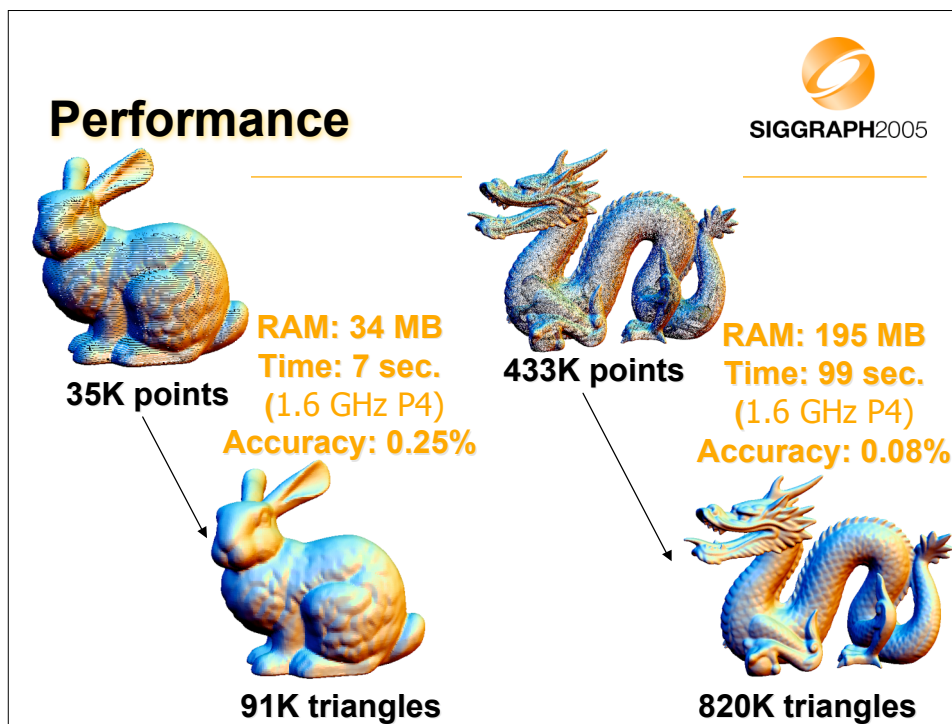
## Computing approximation at $x$



Computing  $f(x)$   
recursively

- **On-the-fly** during
  - polygonization
  - rendering
  - function-based operations
  - ...





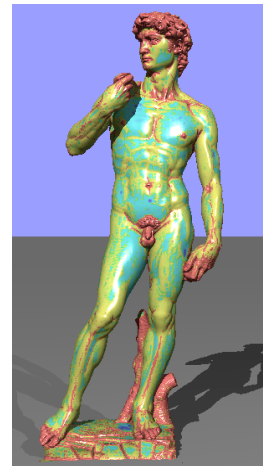


## Multi-level PuO Implicits



SIGGRAPH2005

- Local computations
  - Insensitive to number of points
  - Different local approximations depending on normal statistics
- Local adaptation to shape complexity
- Sensitive to output complexity
- Source code (and more info):  
[www.mpi-sb.mpg.de/~ohtake/mpu\\_implicit/](http://www.mpi-sb.mpg.de/~ohtake/mpu_implicit/)



## Overview



SIGGRAPH2005

- Introduction & Basics
  - Notation, functional approximation
- Multi-level Partition of Unity Implicits
  - Implicit approximation, sharp features, computation, results
- Extensions
  - Other spatial arrangements
  - Faster rendering

## Extensions

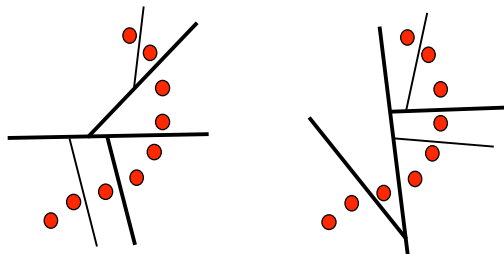


- Other spatial arrangements
  - kD tree instead of Octree
  - BSP tree instead of kD tree, geometry dependent cells
  - Bounding sphere cover
- Faster rendering
  - Blending intersections with polynomials along view rays

## Geometry-dependent cells



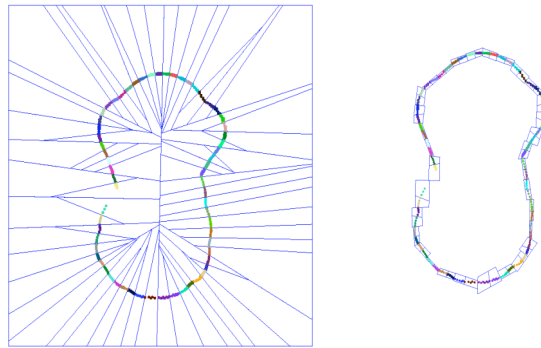
- BSP based on a planarity heuristic
  - Heuristic depends on the co-variances of positions and normals



## Geometry-dependent cells



- BSP based on a planarity heuristic
  - Cells are cropped to tightly bound curve/surface



## Geometry-dependent cells



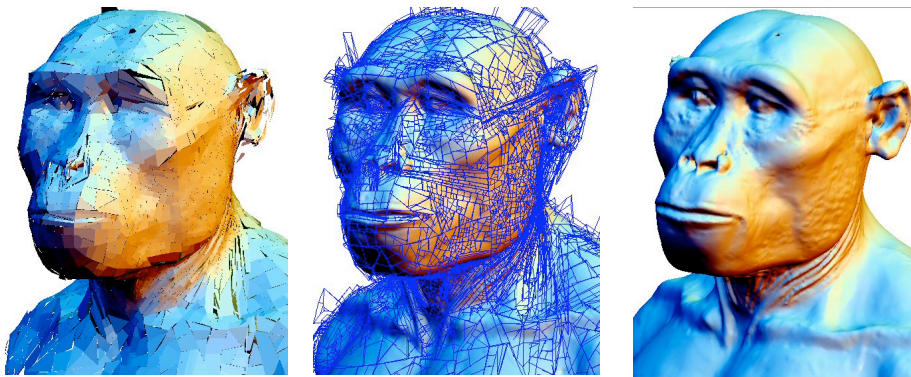
- BSP based on a planarity heuristic
  - In addition, bad aspect ratios of cells are avoided



## Geometry-dependent cells



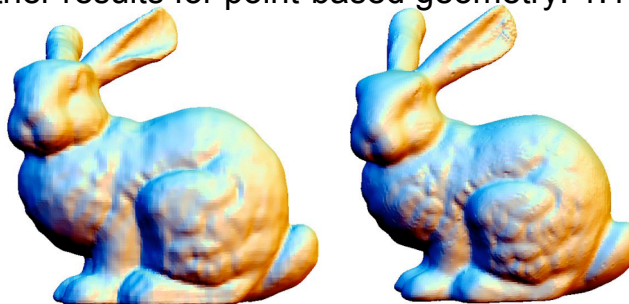
- BSP based on a planarity heuristic



## Geometry dependent cells - results



- Compression 1:25 without entropy coding
  - Other results for point-based geometry: 1:15-1:22



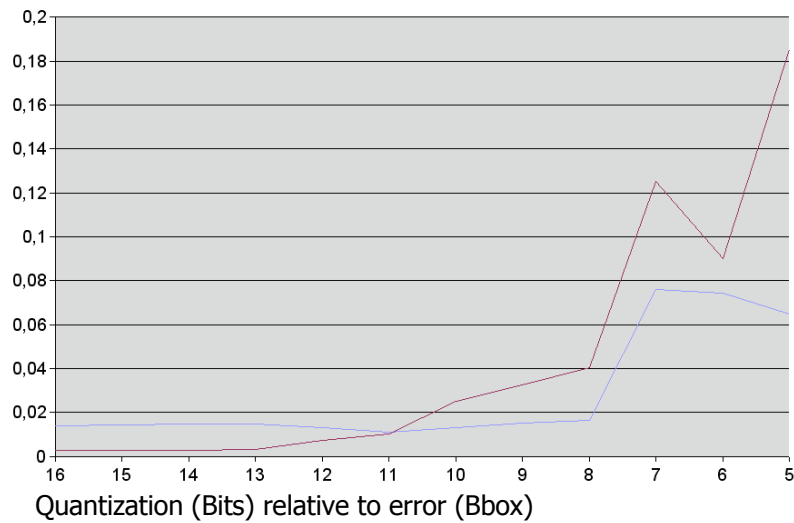
KD-tree 6 bit  
22716 cells, 96 Kbytes  
 $e < 0.0135$

BSP 12 bit  
6269 cells, 55 Kbytes  
 $e < 0.0064$

## Geometry dependent cells - results



SIGGRAPH2005

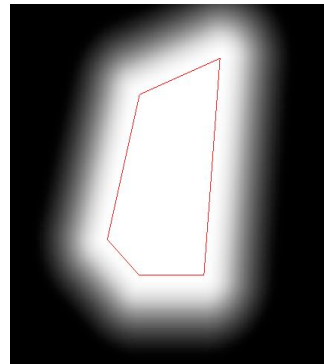


## Geometry dependent cells - problem



SIGGRAPH2005

- Weight functions for blending: ideally distance function to the cell
- Expensive computations

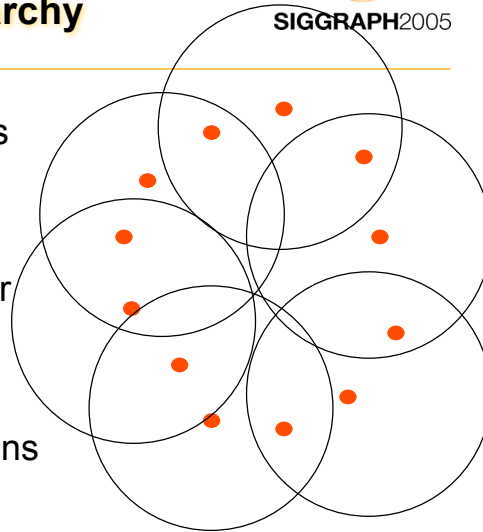


## Bounding volume hierarchy



SIGGRAPH2005

- Cover point with spheres
  - Until all points are multiply covered
- Compute one normal per cell
- Compute local bi-polynomial approximations
  - Convert them to implicits

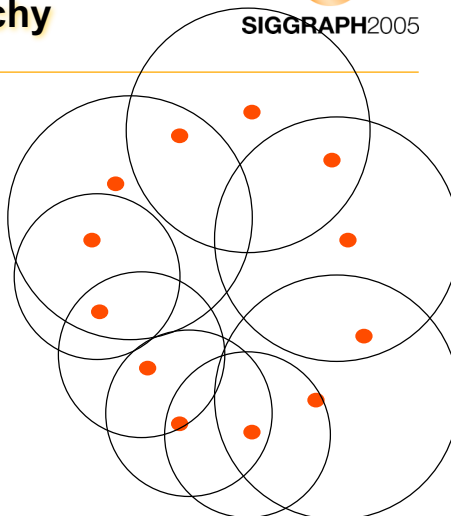


## Bounding volume hierarchy



SIGGRAPH2005

- Multi-level approach
  - Remove cells with large error
  - Re-cover with smaller cells and repeat procedure



## Bounding volume hierarchy



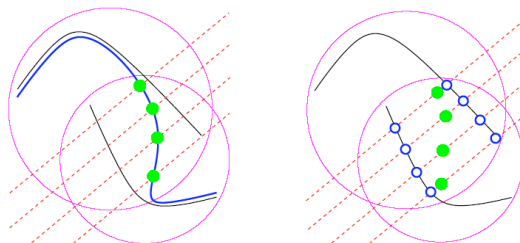
- Covering during the process



## Quick rendering



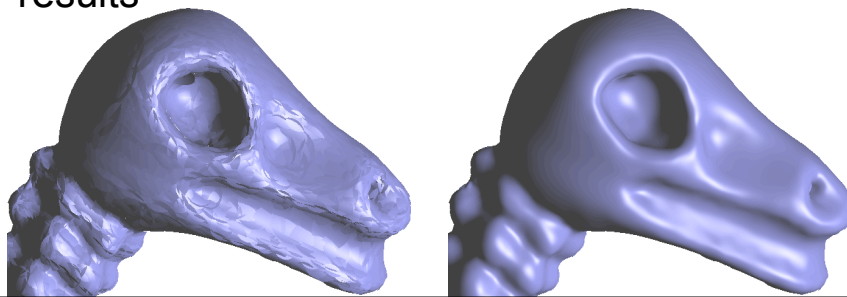
- Right way: Find zero level-set of blended local implicit functions on view rays
  - One dimensional root finding on rational function
- Approximate: Find zero level-set of each local implicit and blend locations
  - One dimensional root finding on polynomials + linear comb.



## Quick rendering



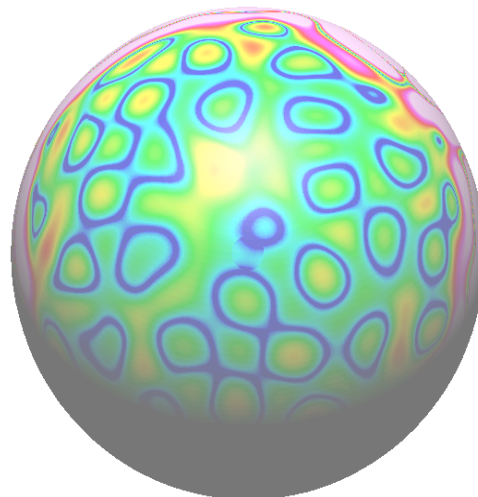
- Using only the first intersection reveals the discontinuous surface representation
- Blending the intersections leads to good results



## Quick rendering



- The absolute error (compared to the 'right' intersections) is overall small
  - Error depends on angle between normal and view ray

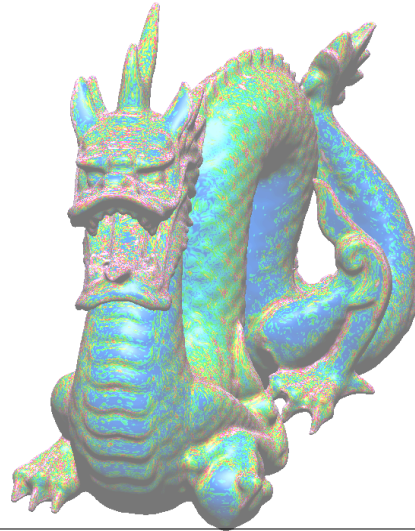




## Quick rendering



- The absolute error (compared to the 'right' intersections) is overall small
  - Error depends on angle between normal and view ray



## Quality rendering



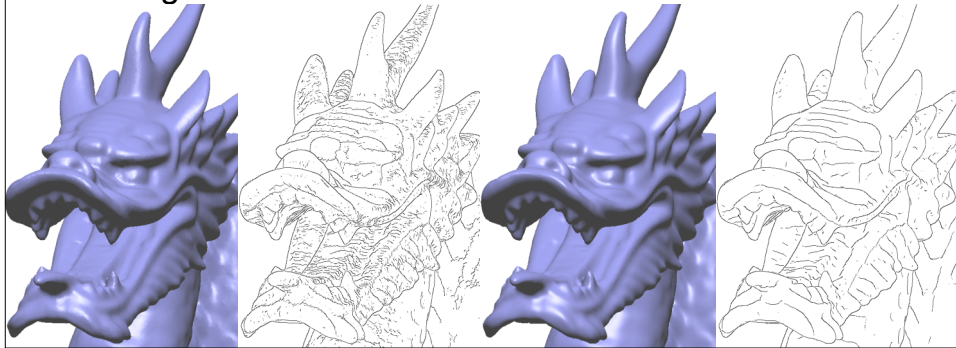
- Quality rendering because of higher order patches
  - Better adaptation to non-monotone curvature
  - No Phong artifacts from linear interpolation of normals
- Example:
  - Gaussian curvature vanishes on a line
  - Compare analytical reflection lines, 50 poly mesh, and 14 quadratic patches



## Quality rendering



- Quality rendering because of higher order patches
  - Higher order differentials available



## Conclusions



- Non-conforming representation
  - Surface is approximated by individual patches
  - Spatial covering defines local implicit approximations, which define the patches
  - Cells could use different types of approximations
  - Blending patches yields surface representation
  - All computations are local
    - Fast, but no global properties guaranteed

# Multi-level Partition of Unity Implicits

Yutaka Ohtake  
MPI Informatik

Alexander Belyaev \*  
MPI Informatik

Marc Alexa  
TU Darmstadt

Greg Turk  
Georgia Tech

Hans-Peter Seidel  
MPI Informatik

## Abstract

We present a new shape representation, the *multi-level partition of unity* implicit surface, that allows us to construct surface models from very large sets of points. There are three key ingredients to our approach: 1) piecewise quadratic functions that capture the local shape of the surface, 2) weighting functions (the partitions of unity) that blend together these local shape functions, and 3) an octree subdivision method that adapts to variations in the complexity of the local shape.

Our approach gives us considerable flexibility in the choice of local shape functions, and in particular we can accurately represent sharp features such as edges and corners by selecting appropriate shape functions. An error-controlled subdivision leads to an adaptive approximation whose time and memory consumption depends on the required accuracy. Due to the separation of local approximation and local blending, the representation is not global and can be created and evaluated rapidly. Because our surfaces are described using implicit functions, operations such as shape blending, offsets, deformations and CSG are simple to perform.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

**Keywords:** partition of unity approximation, error-controlled subdivision, adaptive distance field approximation, implicit modeling.

## 1 Introduction

There are many applications that rely on building accurate models of real-world objects such as sculptures, damaged machine parts, archaeological artifacts, and terrain. Techniques for digitizing objects include laser rangefinding, mechanical touch probes, and computer vision techniques such as depth from stereo. Some of these techniques can yield millions of 3D point locations on the object that is being digitized. Once these points have been collected, it is a non-trivial task to build a surface representation that is faithful to the collected data. Some of the desirable properties of a surface reconstruction method include speed, low memory overhead, the creation of surfaces that approximate rather than interpolate the data (when noise or mis-registration is present), faithful reproduction of sharp features, and robustness in the presence of holes and low sampling density.

In this paper we introduce a new class of implicit models that was specifically designed to meet these requirements for rapidly and accurately creating surfaces from large collections of points. We use the name *Multi-level Partition of Unity* implicits (MPU)

\* Currently with the University of Aizu, Aizu-Wakamatsu, Japan.



Figure 1: The Stanford Lucy, consisting of 14 million points, is reconstructed as an MPU implicit with a 0.01% max-norm approximation accuracy; the left part of the model is colored according to the subdivision level which increases from blue to red. The four models in the back are reconstructed from the point set with increasing approximation error.

because at the heart of our method is a set of weighting functions that sum to one at all points in the domain. Given a set of points  $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_N\}$  sampled from a surface in  $\mathbb{R}^3$ , an MPU implicit  $f: \mathbb{R}^3 \rightarrow \mathbb{R}$  provides an adaptive error-controlled approximation of the signed distance function from the surface. The approximation is accurate near the surface and rough far from the surface. The surface itself is then approximated by the zero level set of the distance function. We assume that the points of  $\mathcal{P}$  are equipped with unit normals  $\mathcal{N} = \{\mathbf{n}_1, \dots, \mathbf{n}_N\}$  that indicate the surface orientation. In practice, these normals can be estimated either from initial scans during the shape acquisition phase or by local least-squares fitting to  $\mathcal{P}$ . We also consider the case when the surface is approximated by a mesh and  $\mathcal{P}$  is the set of mesh vertices. Then the normals  $\mathcal{N}$  are the mesh vertex normals.

To create our implicit representation, we start with a box that bounds the point set and create an octree-based subdivision of this box. At each cell of the octree, a piecewise quadratic function (the *local shape function*) is created that fits the points in the cell. These shape functions act much like a signed distance function, and take on the value zero near the data points and become positive (inside)

or negative (outside) away from the data points. The approximate normals of the points are used to distinguish this inside/outside orientation locally. If the shape function approximation is not accurate enough (doesn't match the points well), the cell is subdivided and the procedure is repeated until a certain accuracy is achieved. Figure 1 shows how the octree-levels adapt according to the relation between local shape complexity and desired accuracy. In locations near the common boundary of two or more cells, the shape functions are blended together according to weights from the *partition of unity* functions. The global implicit function of the surface is given by this partition of unity blending of the local shape approximations at the octree leaves.

MPU implicits are conceptually simple, easy to implement, and are capable of providing a fast, accurate, and adaptive reconstructions of complex shapes from scattered point data containing millions of points. The complexity of the approach is output sensitive in the sense that the creation time and memory consumption depend on the complexity of the reconstructed shape rather than the number of data points. Since MPU implicits can deliver high-accuracy shape approximations, function-based operations such as shape blending, offsets, deformations and CSG can easily be applied. All of these same operations can be performed for data that was originally in a parametric or polygonal form simply by converting these shape descriptions to the MPU representation.

**Previous work.** Implicit shape representations are attractive because they allow a complex shape to be described by one formula, they unify surface and volume modeling, and several complex shape editing operations are easy to perform on such models [Bloomenthal et al. 1997]. On the other hand, traditional pure implicit surface modeling techniques lack local shape control. This drawback has become especially noticeable with the development of modern shape acquisition techniques that can generate data sets consisting of thousands, millions, or even billions of points (see, e.g., [Levoy et al. 2000]). The main advantages of using implicits for shape reconstruction from scattered data are data repairing capabilities and opportunities to edit the resulting objects using standard implicit modeling operations.

Most implicit shape reconstructions from point sets are based on Blinn's idea of blending local implicit primitives [Blinn 1982]. Muraki [1991] uses a linear combination of Gaussian blobs to fit an implicit surface to a point set. Hoppe et al. [1992] locally estimate the signed distance function as the distance to the tangent plane of the closest point. Lim et al. [1995] use blended union of spheres for implicit reconstruction of solids from scattered data. They obtain an initial configuration of spheres from the Delaunay tetrahedralization and a nonlinear optimization is then applied. Bajaj et al. [1995] and Bernardini et al. [1999] combine algebraic fitting with point data triangulation by adaptive  $\alpha$ -shapes [Edelsbrunner and Mücke 1994]. A volumetric approach of Curless and Levoy [1996] introduced for shape reconstruction from range scans is based on estimating the distance function from a reconstructed model. Savchenko et al. [1995], Carr et al. [2001], and Turk and O'Brien [2002] use globally supported radial basis functions (RBFs) while Morse et al. [2001], Kojekine et al. [2003], and Ohtake et al. [2003] employ compactly supported RBFs to reconstruct smooth surfaces from point cloud data. It seems that the state-of-the-art in constructing implicit functions from large sets of scattered points are RBF-based methods [Carr et al. 2001; Dinh et al. 2002; Turk and O'Brien 2002]. While RBF-based methods are especially useful for the repair of incomplete data, they face serious difficulties in accurate reconstruction of sharp features [Dinh et al. 2001], may require a user intervention like choosing an appropriate carrier solid [Kojekine et al. 2003], and can generate extra zero-level sets [Ohtake et al. 2003]. In addition, since RBF solutions are global in nature, processing millions of points seems to be beyond the capabilities of most present day PCs.

The level set method [Zhao and Osher 2002] is another good candidate for reconstructing the signed distance function. However, its current implementation becomes expensive in time and memory if high accuracy reconstruction is required (although this might be improved if adaptive grids are used). Projection-based approaches to shape approximation [Alexa et al. 2001; Fleishman et al. 2003] have the advantage that they are local (i.e. independent of the number of data points) and directly yield a point on the surface. However, the projection step requires the solution of a non-linear moving least squares problem, which makes most practical shape operations expensive.

Our approach can be seen as a blend of several known techniques that, together, result in an attractive method for reconstructing an implicit function. One common RBF technique is to first divide the data domain into several cells so that the data is broken into manageable pieces [Beatson et al. 2000; Schaback and Wendland 2000; Iske 2001; Iske and Levesley 2002; Wendland 2002]. As a particular method for domain decomposition, the partition of unity approach (PU) of Franke and Nielson [1980] has been used as a general FEM method in computational mechanics [Babuška and Osborn 1994] and recently has become popular because it avoids the topological overhead of constructing a mesh [Babuška and Melnik 1997; Griebel and Schweitzer 2000; Griebel and Schweitzer 2002]. Our strategy for avoiding extra zero level sets is reminiscent of [Moore and Warren 1991; Warren 1992], where an adaptive and recursive volumetric subdivision was used. One could view our MPU representations as being similar to adaptively sampled distance fields [Friskien et al. 2000], with the difference that the MPU approach uses continuous rather than sampled functions.

When used with an appropriate choice of local shape approximations, our approach has the following attractive features: the ability to create high quality implicit surfaces from very large point datasets, the accurate reconstruction of sharp features, and fast and easy local shape access.

## 2 Partition of Unity Approach

The partition of unity approach is typically used to integrate locally defined approximants into a global approximation. Important properties such as the maximum error and convergence order are inherited from the local behavior. The basic idea of the partition of unity approach is to break the data domain into several pieces, approximate the data in each subdomain separately, and then blend the local solutions together using smooth, local weights that sum up to one everywhere on the domain.

More specifically, consider a bounded domain  $\Omega$  in a Euclidean space (we will work in 3D) and a set of nonnegative compactly supported functions  $\{\varphi_i\}$  such that

$$\sum_i \varphi_i \equiv 1 \text{ on } \Omega.$$

Let us associate a local approximation set of functions  $V_i$  with each subdomain  $\text{supp}(\varphi_i)$ . Now an approximation of a function  $f(\mathbf{x})$  defined on  $\Omega$  is given by

$$f(\mathbf{x}) \approx \sum_i \varphi_i(\mathbf{x}) Q_i(\mathbf{x}), \quad (1)$$

where  $Q_i \in V_i$ .

Given a set of nonnegative compactly supported functions  $\{w_i(\mathbf{x})\}$  such that

$$\Omega \subset \bigcup_i \text{supp}(w_i),$$

partition of unity functions  $\{\varphi_i\}$  can be generated by

$$\varphi_i(\mathbf{x}) = \frac{w_i(\mathbf{x})}{\sum_{j=1}^n w_j(\mathbf{x})}. \quad (2)$$

Approximation by means of Eqs. 1 and 2 constitutes the core of the partition of unity finite element methods [Babuška and Osborn 1994]. They resemble the Modified Shepard's method of Franke

and Nielson [1980] (see also [Renka 1988]), where polynomial local approximations  $Q_i(\mathbf{x})$  are used in combination with “inverse-distance” singular weights  $\{w_i(\mathbf{x})\}$  for interpolation purposes.

Given a set of scattered points  $\mathcal{P}$  equipped with normals  $\mathcal{N}$ , we approximate the signed distance function  $f(\mathbf{x})$  from  $\mathcal{P}$ . In contrast to the approaches mentioned above, we introduce an adaptive procedure for generating the subdivision and problem-specific approximation sets. First, we use an octree-based adaptive space subdivision of  $\Omega$ . This allows us to control the error of the approximation while adapting the complexity of the representation to the complexity of the shape (see Section 3). Second, we use *piecewise* quadratic functions resulting from Boolean operations for the accurate representation of sharp features. The classification of local shapes and appropriate approximation sets are discussed in Section 4.

For approximation purposes we use the quadratic B-spline  $b(t)$  to generate weight functions

$$w_i(\mathbf{x}) = b\left(\frac{3|\mathbf{x} - \mathbf{c}_i|}{2R_i}\right) \quad (3)$$

centered at  $\mathbf{c}_i$  and having a spherical support of radius  $R_i$ .

If an interpolation of  $\mathcal{P}$  is required, we use the inverse-distance singular weights [Franke and Nielson 1980; Renka 1988]

$$w_i(\mathbf{x}) = \left[ \frac{(R_i - |\mathbf{x} - \mathbf{c}_i|)_+}{R_i |\mathbf{x} - \mathbf{c}_i|} \right]^2, \text{ where } (a)_+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

### 3 Adaptive Octree-based Approximation

The algorithm for constructing an MPU implicit is driven by repeated subdivision of the region of space that is occupied by the input set of points. First, the points in  $\mathcal{P}$  are rescaled so that an axis-aligned bounding cube has a unit-length main diagonal. We then apply an adaptive octree-based subdivision to the bounding cube. Consider a cubic cell that was generated during the subdivision process, and let  $\mathbf{c}$  be the center and  $d$  the length of the main diagonal of the cell.

We define the support radius  $R$  for the weight function (3) for the cell to be proportional to its diagonal:

$$R = \alpha d. \quad (5)$$

We typically use  $\alpha = 0.75$ . A larger value for  $\alpha$  yields better (smoother) interpolation and approximation results at the expense of computation time. Time complexity is roughly quadratic in  $\alpha$ . Figure 11 illustrates the effect of  $\alpha$ , especially for the accurate approximation of the distance function away from the zero level set.

For each cell generated during the subdivision process, a local shape function  $Q(x)$  is built using a least-squares fitting procedure, as shown in the left drawing of Figure 2.

Sometimes (especially if the density of  $\mathcal{P}$  is not uniform) the ball of radius  $R$  for a cell does not contain enough points for a robust estimation of  $Q(x)$ . If the number of points is less than  $N_{\min}$  (in our implementation we set  $N_{\min} = 15$ ), a ball with increased radius  $\hat{R}$  is determined that contains at least  $N_{\min}$  points. This is done by starting with  $\hat{R} = R$  and then iterating

$$\hat{R} = \hat{R} + \lambda R \quad (6)$$

until the ball contains the minimum number of points (in our implementation we set  $\lambda = 0.1$ ). Now the points enclosed by the ball of radius  $\hat{R}$  are used to estimate a local shape function  $Q(\mathbf{x})$ , as demonstrated in the right drawing of Figure 2.

We use a *kd*-tree partitioning for efficient solving of these range searching problems.

If the ball around an octree cell with initial radius  $R = \alpha d$  is empty, an approximation of the distance function is computed as explained above and it will not be subdivided further. Otherwise, a local max-norm approximation error is estimated according to the Taubin distance [Taubin 1991]

$$\varepsilon = \max_{|\mathbf{p}_i - \mathbf{c}| < R} |Q(\mathbf{p}_i)| / |\nabla Q(\mathbf{p}_i)|. \quad (7)$$

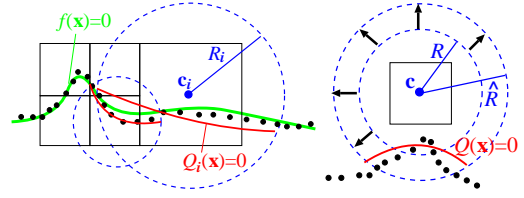


Figure 2: Left: adaptive subdivision coupled with least-squares fitting. Right: enlarging the spherical domain for the local approximation to make it more robust.

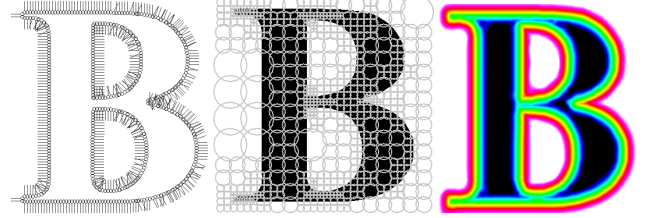


Figure 3: Left: a set of points equipped with normals. Middle: the circles (2D balls) corresponding to the adaptive subdivision are shown here at 50% of their real size. Right: the distance function is reconstructed, and the zero level set is located between the yellow and green bands.

If  $\varepsilon$  is greater than a user-specified threshold  $\varepsilon_0$ , the cell is subdivided and the fitting process is performed for the child cells.

Figure 3 demonstrates how our adaptive subdivision scheme works in 2D.

The following pseudocode describes a recursive procedure for assembling an MPU approximation at point  $\mathbf{x}$  with precision  $\varepsilon_0$ .

```

MPUapprox( $\mathbf{x}, \varepsilon_0$ )
  if ( $|\mathbf{x} - \mathbf{c}_i| > R_i$ ) then return;
  if ( $Q_i$  is not created yet) then
    Create  $Q_i$  and compute  $\varepsilon_i$ ;
  if ( $\varepsilon_i > \varepsilon_0$ ) then
    if (No childs) then Create childs;
    for each child
      child  $\rightarrow$  MPUapprox( $\mathbf{x}, \varepsilon_0$ );
  else
     $S_{wQ} = S_{wQ} + w_i(\mathbf{x}) * Q_i(\mathbf{x})$ ;
     $S_w = S_w + w_i(\mathbf{x})$ ;

EvaluateMPUapprox( $\mathbf{x}, \varepsilon_0$ )
   $S_{wQ} = S_w = 0$ ;
  root  $\rightarrow$  MPUapprox( $\mathbf{x}, \varepsilon_0$ );
  return  $S_{wQ}/S_w$ ;

```

Here,  $S_w$  and  $S_{wQ}$  denote  $\sum w_i(\mathbf{x})$  and  $\sum Q_i(\mathbf{x})w_i(\mathbf{x})$ , respectively, see Eqs. 1 and 2.

This procedure is easy to implement. We hope that this will make our approach and its implementation accessible to a wide range of users.

Note that we abandon the local approximations that are constructed at the non-leaf cells of the octree. This allows us to use different local approximations of the distance function far from  $\mathcal{P}$  and near to  $\mathcal{P}$ , as well as specific sharp feature approximations, without compensating for the effects of coarse approximations. Inheriting coarse approximations would also require us to counteract already generated zero-sets in empty balls. In addition, avoiding coarse approximations saves memory and results in faster evaluation of the implicit function.

### 4 Estimating Local Shape Functions

Our local fitting strategy depends on the number of points in the ball of a given cell and the distribution of normals of those points. At a given cell we use the most appropriate one of these three local approximations:

- a general 3D quadric,
- a bivariate quadratic polynomial in local coordinates,
- a piecewise quadric surface that fits an edge or a corner.

Roughly speaking, (a) is used to approximate larger parts of the surface, which could be unbounded or contain more than one sheet, (b) is used to approximate a local smooth patch, and (c) is employed



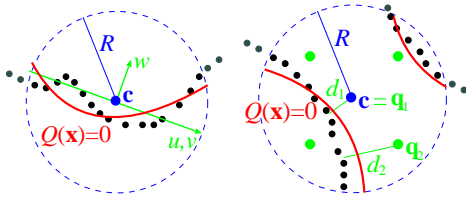


Figure 4: Left: fitting a bivariate quadratic polynomial. Right: local fit of a general quadric; auxiliary points located at the cell center and corners are used in order to achieve an accurate approximation of the distance function.

to reconstruct sharp features. Actually (c) consists of a number of tests (an edge test and corner tests) in order to determine the type of approximation surface that should be used.

A few simple tests are performed to select from among the three types of local shape functions. If there are more than  $2N_{\min}$  points in the local ball, we use a function of type (a) or (b). An average normal direction is computed for the points and if the maximum deviation of normals to the average is more than  $\pi/2$  then we fit with (a), otherwise we use type (b). If there are fewer than  $2N_{\min}$  points associated with a cell, more detailed checks are made to see if an edge or corner is present, and details of this are given below. Why don't we look for sharp features when there are more than  $2N_{\min}$  points? Because the sharp feature detection adds computational complexity and the octree subdivision procedure takes care of this, anyway. Should the surface actually contain a corner or edge near such a cell, then the quality-of-fit measure (7) will cause the cell to be divided, and the sharp feature will be fit by one or more child cells.

In the following sub-sections we will describe each of the three local shape functions in more detail. The notation we use in these sections is as follows. Let  $\mathbf{c}$  be the center of a subdivision cubic cell where we want to construct a local shape function  $Q(\mathbf{x})$ . We denote by  $\mathcal{P}'$  the points of  $\mathcal{P}$  that are inside the ball of the cell. Let  $\mathbf{n}$  be a unit normal vector assigned to  $\mathbf{c}$ . This normal  $\mathbf{n}$  is computed from the normalized weighted arithmetic mean of the normals of  $\mathcal{P}'$  taken with the weights defined by (3). Let  $\theta$  be the maximal angle between  $\mathbf{n}$  and the normals  $\mathcal{N}'$  assigned to the points of  $\mathcal{P}'$ .

**(a) Local fit of a general quadric.** If

$$|\mathcal{P}'| > 2N_{\min} \quad \text{and} \quad \theta \geq \pi/2$$

a 3D quadratic surface is fitted. Usually this corresponds to a situation sketched in the right drawing of Figure 4. A local shape function is given by

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c \quad (8)$$

where  $\mathbf{A}$  is a symmetric  $3 \times 3$  matrix,  $\mathbf{b}$  is a vector of three components, and  $c$  is a scalar. In order to determine the unknowns  $\mathbf{A}$ ,  $\mathbf{b}$ , and  $c$  we make use of auxiliary points  $\{\mathbf{q}_i\}$  to help orient the local shape function. These points are chosen as the corners and the center of the subdivision cell, as demonstrated in the right picture of Figure 4.

Each auxiliary point  $\mathbf{q}$  is tested for whether it can be used to obtain a reliable estimate of the signed distance function. For each  $\mathbf{q}$ , its six nearest neighbors  $\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \dots, \mathbf{p}^{(6)}$  from  $\mathcal{P}'$  are found and the scalar products

$$\mathbf{n}^{(i)} \cdot (\mathbf{q} - \mathbf{p}^{(i)}), \quad i = 1, 2, \dots, 6, \quad (9)$$

are computed. If not all the scalar products have the same sign,  $\mathbf{q}$  is removed from the set of auxiliary points. The geometric idea behind this test is explained by the left drawing of Figure 5.

If the set of remaining auxiliary points is empty, the cell is subdivided.

For each remaining auxiliary point  $\mathbf{q}$ , an average distance  $d$ , the arithmetic mean of the scalar products (9), is computed

$$d = \frac{1}{6} \sum_{i=1}^6 \mathbf{n}^{(i)} \cdot (\mathbf{q} - \mathbf{p}^{(i)}). \quad (10)$$

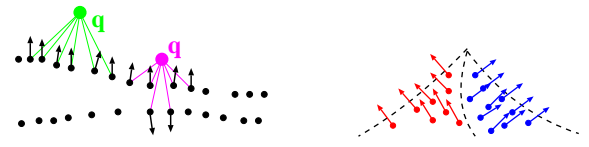


Figure 5: Left: Testing whether an auxiliary point  $\mathbf{q}$  can be used to obtain a reliable estimate of the signed distance function. The green  $\mathbf{q}$  is reliable, but the magenta  $\mathbf{q}$  is not. Right: Detection of sharp features is done by clustering of point normals.

Finally the unknowns in (8) are found by minimizing

$$\frac{1}{\sum w(\mathbf{p}_i)} \sum_{\mathbf{p}_i \in \mathcal{P}'} w(\mathbf{p}_i) Q(\mathbf{p}_i)^2 + \frac{1}{m} \sum_{i=1}^m (Q(\mathbf{q}_i) - d_i)^2, \quad (11)$$

where  $m$  is the number of remaining points  $\mathbf{q}$ .

**(b) Local fit of a bivariate quadratic polynomial.** If

$$|\mathcal{P}'| > 2N_{\min} \quad \text{and} \quad \theta < \pi/2$$

a bivariate quadratic polynomial is locally fitted. Let us introduce local coordinates  $(u, v, w)$  with the origin of coordinates at  $\mathbf{c}$  such that the plane  $(u, v)$  is orthogonal to  $\mathbf{n}$  and the positive direction of  $w$  coincides with the direction of  $\mathbf{n}$ . A quadratic shape function at  $\mathbf{c}$  is given by

$$Q(\mathbf{x}) = w - (Au^2 + 2Buv + Cv^2 + Du + Ev + F), \quad (12)$$

where  $(v, u, w)$  are the coordinates of  $\mathbf{x}$  in the new coordinate system. The unknown coefficients  $A, B, C, D, E$ , and  $F$  are determined by minimizing

$$\sum_{\mathbf{p}_i \in \mathcal{P}'} w(\mathbf{p}_i) Q(\mathbf{p}_i)^2. \quad (13)$$

The left drawing of Figure 4 illustrates the geometric idea behind local fitting of a bivariate quadratic polynomial.

**(c) Local approximation of edges and corners.** The quadratic functions (12) and (8) considered above are not capable of accurately capturing sharp edges and corners. If there are just a few points associated with a cell ( $|\mathcal{P}'| \leq 2N_{\min}$ ), we try to fit a *piecewise* quadratic function instead of a quadratic approximation.

We perform automatic recognition of edges and corners using a simple but effective procedure proposed by Kobbelt et al. [2001]. The idea is based on clustering of normals, as demonstrated by the right drawing of Figure 5.

Following [Kobbelt et al. 2001] we assume that the surface has a sharp feature if

$$\min_{i,j} (\mathbf{n}_i \cdot \mathbf{n}_j) < 0.9. \quad (14)$$

If (14) is not satisfied, we go to (b) and local bivariate polynomial (12) is fitted to  $\mathcal{P}'$ . Otherwise we check whether the detected feature is a corner. We consider  $\mathbf{n}_3 = \mathbf{n}_1 \times \mathbf{n}_2$ , the normal vector to the plane determined by the normals  $\mathbf{n}_1$  and  $\mathbf{n}_2$  enclosing the maximal angle. If the deviation of the normals  $\mathbf{n}_i$  from the plane is sufficiently large

$$\max_i |\mathbf{n}_i \cdot \mathbf{n}_3| > 0.7 \quad (15)$$

the feature is recognized as a corner.

If (14) is satisfied and (15) is not, we expect the surface to have an edge and subdivide the set of normals  $\{\mathbf{n}_i\}$  into two clusters according to their angles with  $\mathbf{n}_1$  and  $\mathbf{n}_2$  (two spherical Voronoi subsets corresponding to  $\mathbf{n}_1$  and  $\mathbf{n}_2$ ). Denote by  $\mathcal{P}'_1$  and  $\mathcal{P}'_2$ ,  $\mathcal{P}' = \mathcal{P}'_1 \cup \mathcal{P}'_2$ , two subsets corresponding to the clusters of the normals. Now the quadratic fit procedure is applied separately to  $\mathcal{P}'_1$  and  $\mathcal{P}'_2$  and a non-smooth local shape function approximation  $\mathcal{P}'$  is constructed via the max/min Boolean operations of Ricci [1973].

If (14) and (15) are satisfied, we subdivide  $\mathcal{N}'$  into three sets. First  $\mathcal{N}'_1$  and  $\mathcal{N}'_2$  are constructed as above. Next we check whether  $|\mathbf{n}_{1,2} \cdot \mathbf{n}_i| < |\mathbf{n}_3 \cdot \mathbf{n}_i|$  and add point  $\mathbf{p}_i$  to the third cluster if the inequality is satisfied.

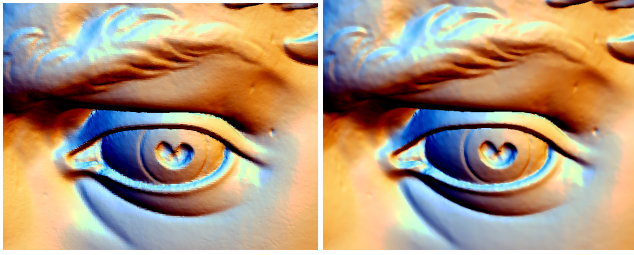


Figure 6: Left: Eye from Stanford’s reconstruction of Michelangelo’s David (scanned at 1mm resolution). Right: The eye is reconstructed as an MPU implicit with relative accuracy  $\epsilon_0 = 10^{-4}$ .

We also treat separately corners of degree four: test (14) is applied to the normals of the third cluster and if (14) is satisfied, the cluster is subdivided into two pieces. If the resulting four clusters of normals correspond to either a convex or concave corner, it is reconstructed via Boolean operations. Otherwise, we go to (a) and a general quadric (8) is fitted to  $\mathcal{P}'$ .

If the number of points used to estimate the coefficients of bi-variate quadratic polynomial (12) is less than six, we set all the unknown coefficients in (12) equal to zero.

Given the above approach, more complex types of sharp features (for example, a saddle corner of degree 4) are approximated by smooth functions. Notice however that “generic” sharp features are obtained from the intersections of two or three surfaces, and therefore consist of edges and corners of at most degree 3. It is not a generic event for four smooth surfaces to intersect at one point.

## 5 Visualization

Conventional techniques for visualizing implicit models include polygonization (isosurface extraction), ray tracing, and volume rendering. From among these various visualization methods, we use Bloomenthal’s polygonizer [Bloomenthal 1994] because of its nice continuation properties and the Hart sphere tracing method [Hart 1996]. These two methods both work well using the approximate distance functions of MPU implicits.

If our goal is to create a polygonal mesh, we can save time and memory by computing MPU approximations on the fly during the polygonization process. We have found that an approximation accuracy of  $\epsilon_0 = 10^{-4}$  (that is, 0.01% of the length of the main diagonal of the bounding box) is quite sufficient for the reconstruction of fine features, as demonstrated by Figure 6.

If a non-adaptive surface extraction routine is used with an implicit model that has sharp features, a fine sampling density is required to capture these features. An example of this can be seen in the top and bottom left images of Figure 7. An alternative is to use adaptive sampling and remeshing strategies such as those in [Kobbelt et al. 2001; Ohtake and Belyaev 2002; Ju et al. 2002]. We find it attractive to combine a low resolution Bloomenthal polygonization with a postprocessing mesh optimization technique developed in [Ohtake and Belyaev 2002], as shown in the top middle, top right, and bottom middle images of Figure 7.

Even higher quality rendering can be achieved using ray tracing techniques. The sphere tracing method of Hart [1996] works well together with MPU implicits since it uses the distance function representation and it is quite capable of rendering implicit models with sharp features. The bottom right image of Figure 7 shows an MPU implicit model of the fandisk model that was rendered with sphere tracing. The left image of Figure 8 shows sphere tracing of a more complex implicit model. This model was generated from a function-based operation (subtraction) applied to the dragon and the David model, both represented as MPU implicits. Notice how well the sharp features are reconstructed and rendered.

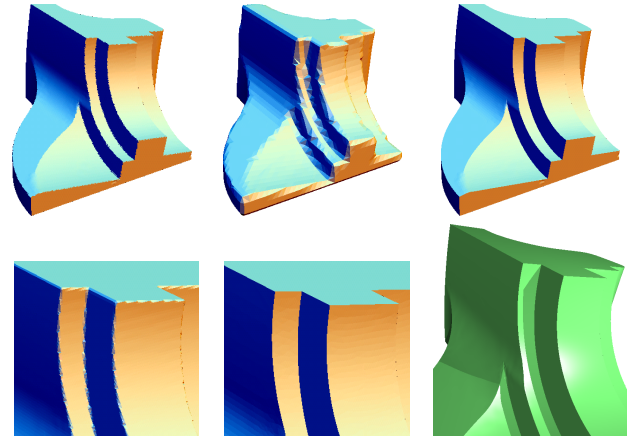


Figure 7: Visualization of the fandisk model implicitized with MPU. Top left: Bloomenthal polygonization was used; in spite of sufficiently high polygonization resolution (200K triangles) one can notice small aliasing defects along sharp features. Top middle: a low resolution polygonization is applied. Top right: an optimized mesh (17K triangles) is obtained from the low resolution mesh, the sharp features are accurately reconstructed. Bottom left: a magnified part of the high resolution mesh. Bottom middle: the same part of the optimized mesh. Bottom right: a part of the model rendered using Hart sphere tracing.

## 6 Results & Applications

In this section we discuss results and applications of approximating or interpolating MPU implicits for surface reconstruction from range scans and incomplete point data, and function-based operations.

**Approximation and Interpolation.** Most of the illustrations in this work have been generated using approximating MPU implicits as described in Section 4. However, our MPU approach can also be adapted to exact data point interpolation if we use a local interpolation method such as Franke and Nielson’s singular weights (4) instead of (3).

Since MPU with (4) requires a deeper octree-based subdivision (every nonempty subdivision cell contains only one point of  $\mathcal{P}$ ), our interpolation procedure requires more memory resources than the approximation one. For interpolation with singular weights (4) the subdivision stops only when all of points of  $\mathcal{P}$  have been placed in their own cells. The ball around a nonempty octree cell is centered at the interpolated point inside the cell. We set  $\alpha = 1.25$  in (5) in order to ensure that we cover the bounding box by the balls around the octree cells.

For each cell containing one interpolated point  $\mathbf{p}$  of  $\mathcal{P}$ , its local shape function  $Q(\mathbf{x})$  is defined by (12), where the origin of coordinates of local coordinate system  $(u, v, w)$  is located at  $\mathbf{p}$  (hence  $F = 0$  in (12)) and the positive direction of  $w$  coincides with the direction of the averaged normal at  $\mathbf{p}$ . We don’t use the normal of  $\mathcal{N}$  assigned to  $\mathbf{p}$  because of stability problems common for Hermite-like interpolation schemes. The unknown coefficients are determined by minimizing quadratic energy (13) with  $\mathbf{c} = \mathbf{p}$ . Now (1) interpolates  $\mathcal{P}$  because partition of unity functions  $\{\varphi_i(\cdot)\}$  defined by (2) satisfy

$$\varphi_i(\mathbf{p}_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad \text{and} \quad \nabla \varphi_i(\mathbf{p}_j) = 0.$$

The right image of Figure 8 shows results of applying the MPU interpolation and shape modeling operations (Boolean subtraction, twisting) to the Stanford Buddha model.

**Reconstruction from incomplete data.** Reconstruction from scattered point data with MPU implicits is robust with respect to variations of point density, as demonstrated in Figure 9.

**Reconstruction from range scans.** MPU implicits can be used to reconstruct 3D models from a collection of range scans. We have

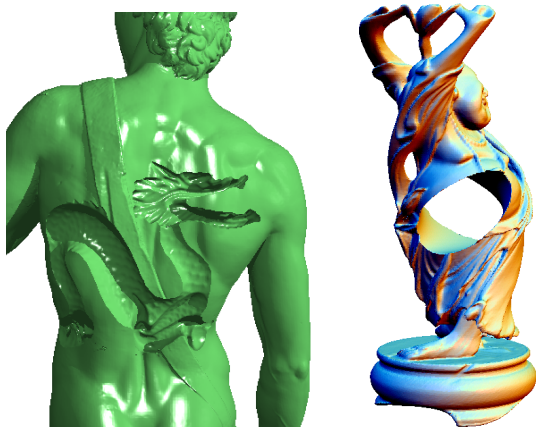


Figure 8: CSG operations applied on MPU implicits. Left: sphere tracing of the subtraction of two MPU approximations. Right: boolean subtraction and twisting operations are applied to interpolating MPU implicits.

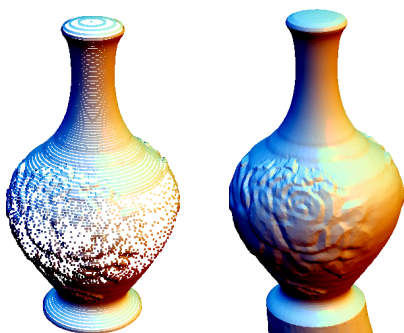


Figure 9: Reconstruction from a scattered point dataset with non-uniform density of points.

found that if several range scans overlap, better results are obtained if we take into account per-point measurement confidences during the reconstruction process. If we treat all points the same, artifacts can arise. If the accuracy threshold (7) is small, the MPU implicit approximating the scan points can have wrinkles in the overlapping regions. On the other hand, if (7) is not small enough, the MPU implicit does not capture the fine geometric details of the scanned model. In practice, a given position on the object can be measured more accurately from some scanning directions than from others. This notion of using confidence during surface reconstruction was advocated in [Turk and Levoy 1994; Curless and Levoy 1996].

Consider a collection of points from range data. Assume that each point  $\mathbf{p}_i$  is assigned a confidence weight  $c_i$ ,  $c_i \in [0, 1]$ , that were computed based on scanning information according to the rules suggested in [Curless n. d.]. Now the MPU reconstruction process described in previous sections is enhanced by the modifications given below.

- For a better estimation of local shape function  $Q(\mathbf{x})$ , if the sum of the confidence measures of the points inside the ball is less than  $N_{\min}$  then the radius growth rule (6) is applied repeatedly until the sum is above this threshold.
- Instead of (7), a weighted accuracy measure is used:  

$$\varepsilon = \max_i c_i Q(\mathbf{p}_i) / |\nabla Q(\mathbf{p}_i)|.$$
- The unit normal vector  $\mathbf{n}$  of the base plane  $(u, v)$  used to fit the bivariate quadratic polynomial (12) is obtained by averaging the neighboring normals with weights  $c_i w(\mathbf{p}_i)$ .
- Weights  $\{c_i w(\mathbf{p}_i)\}$  are used in (13) and (11) instead of  $\{w(\mathbf{p}_i)\}$ .
- The normals in (10) are taken with the confidence weights assigned to their corresponding points.

Figure 10 demonstrates the MPU reconstruction of the Stanford

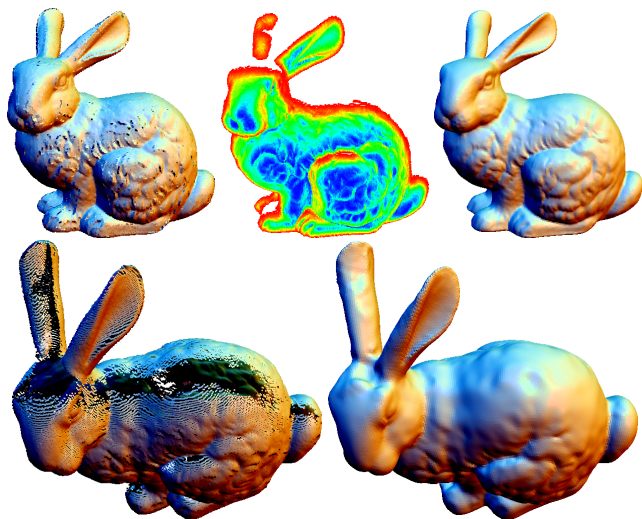


Figure 10: Reconstruction of Stanford bunny from range data. Top left: bunny scan data is rendered as a cloud of points, (all ten original range scans are used); defects caused by low accuracy of some points and normals are clearly visible. Top middle: a side range image of bunny is colored according to the confidence measure. Top right: bunny is reconstructed as an MPU implicit. Bottom left: only six range scans of the bunny scan data are rendered (an example of incomplete data). Bottom right: an MPU implicit bunny from six scans.

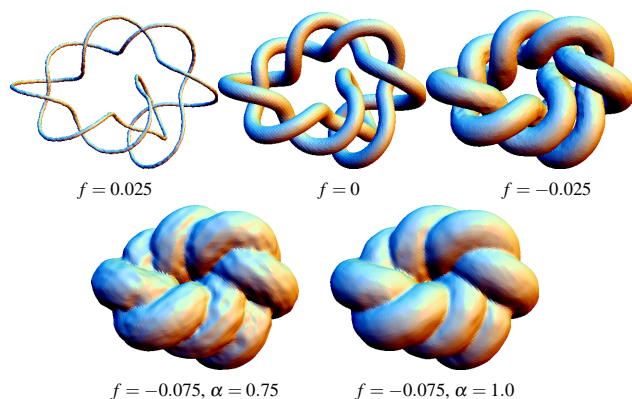


Figure 11: Offsetting of a knot model. The distance function to the knot is approximated by  $w = f(x, y, z)$ . The first four models were generated with  $\alpha = 0.75$ . For the last model  $\alpha = 1$  was used and a higher quality offsetting was produced.

bunny from the original range scans. In one case we have used only six scans, and in the other case we have used the full ten range scans. Notice the ability of the MPU method to repair missed data.

**Function-based shape modeling operations.** Using MPU implicits allow us to extend the power of function-based shape modeling operations [Bloomenthal et al. 1997] to point set surfaces. Given several MPU functions defined over the same bounding box and having possibly different octree structures, at each point of the box we evaluate the value of the result of applying functional operations to the functions. Then the level sets of the resulting function are visualized via a polygonization or ray tracing.

An example of a CSG operation applied to two large and complex point set surfaces was already demonstrated in Figure 8. Results of offsetting, smooth blending, morphing, and twisting operations [HyperFun: F-rep Library n. d.], [Pasko and Savchenko 1994] are shown in Figures 11, 12, 13, 14 and the right image of Figure 8. In particular, Figures 13 and 14 demonstrate a linear morphing of two implicit models.<sup>1</sup>

<sup>1</sup>The linear morphing of implicit models  $w = f(x, y, z)$  and  $w = g(x, y, z)$  is an implicit model defined by  $w = (1 - t)f(x, y, z) + tg(x, y, z)$ .



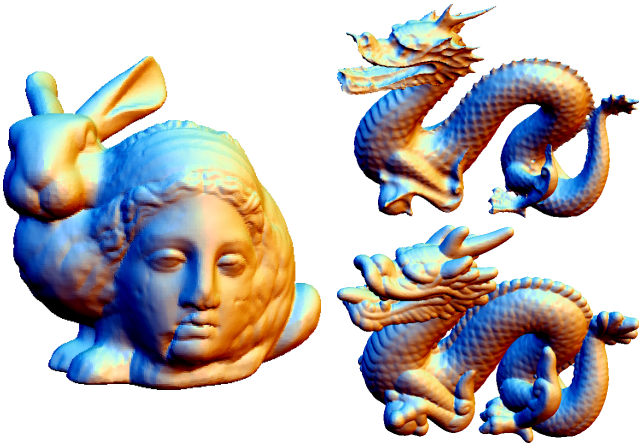


Figure 12: Left: Smooth blending of the Stanford bunny and Cyberware Igea models. Right: offsetting of the Stanford dragon model;  $f = 0.0075$  (top) and  $f = -0.01$  (bottom);  $\alpha = 0.75$ .

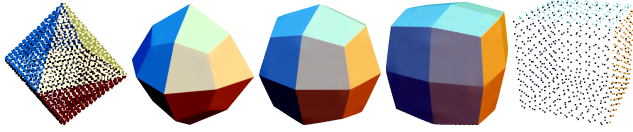


Figure 13: Linear blending of octahedron and cube.

## 7 Discussion

This paper describes a new implicit surface representation based on local shape functions, partitions of unity, and an octree hierarchy. Strengths of the Multi-Level Partition of Unity formulation include:

- Fast surface reconstruction and rendering.
- Representation of sharp features.
- Reconstruction from incomplete data.
- Choice of either approximation or interpolation of the data and the ability to adaptively vary the approximation accuracy.

Given a point set model processed by the MPU method with a specified accuracy, the computational time and memory usage depends on the geometric complexity of the model: the higher the geometric complexity, the deeper the octree is subdivided. This is clearly demonstrated by Figure 1 where the reconstruction of fine features requires a deeper subdivision.

Table 1 presents RAM memory usage and computational time measurements for simultaneous generating and polygonizing various point set models. Note that our method is quite fast. Our experiments with state-of-the-art RBF-based 3D surface reconstruction techniques such as FastRBF [Carr et al. 2001] and others suggest that the MPU method is considerably faster than these other techniques.<sup>2</sup>

Model	Number of points	Relative error	Peak RAM	Number of triangles	Comp. time
Bunny	34,611	$2.5 \times 10^{-3}$	34 MB	91,104	0:07
Bunny scans	362,272	$1.0 \times 10^{-3}$	110 MB	219,676	1:46
Dragon	433,375	$8.0 \times 10^{-4}$	195 MB	819,704	1:39
Buddha	543,625	0.0	442 MB	648,332	6:53
David (2mm)	4,124,454	$1.0 \times 10^{-4}$	810 MB	1,296,522	10:33

Table 1: Memory and computational time measurements for generation + polygonization of MPU implicits for various point set models. Computations were performed on a 1.6GHz Mobile Pentium 4 with 1GB RAM, and timings are listed as minutes:seconds.

<sup>2</sup> Comparing the results of Table 1 with those of Table 2 in [Carr et al. 2001] one can find that the MPU method is 20-30 times faster than the FastRBF technique [Carr et al. 2001].



Figure 14: Linear morphing of two head models approximated by MPU implicits, Max Planck Head and Head of Michelangelo's David.

Notice that processing time for the Buddha model is more than three times longer than that for the dragon model which has a similar size. This is because we reconstruct the Buddha by the MPU interpolation which requires a deeper subdivision and wider support for the corresponding partition of unity functions.

Because of its local nature, the MPU method is more sensitive to the quality of input data, especially the field of normals, to compare with the approximation and interpolation techniques based on globally-supported RBFs [Carr et al. 2001; Turk and O'Brien 2002]. Nevertheless, according to our experiments, the MPU method is sufficient for accurate shape reconstruction from a wide variety of data sets. The parameters in our current implementation of the MPU approach are adjusted for processing typical outputs of laser scanner devices. We believe that the parameter modifications needed for different classes of input are fairly intuitive in order to handle scattered data of lower (higher) quality at the expense of lower (higher) computational speed.

Unlike the crust approach [Amenta et al. 1998], the MPU method is not supported by rigorous results guaranteeing correct reconstruction of input data satisfying certain properties described mathematically. It is a price we pay for a high speed of our method.

We would like also to stress here that our method is not an RBF technique. RBF is a global approximation/interpolation method because of its global variational nature. Our method is a local one. We make use of *two* functions, the partition-of-unity weights and the local piecewise quadratic approximation functions, which is different than the single function used by an RBF approach. This two-function approach gives benefits such as sharp feature reconstruction that, to date, have not been possible using RBFs.

Smoothness properties of the MPU implicits are determined by those of weight functions (3). Choosing smoother weight functions will produce smoother MPU implicits.

Similar to other implicit function shape representation schemes, the MPU implicits are not capable of representing correctly surfaces with boundaries.

We see a number of opportunities to improve our approach. Other estimation of the distance function might be beneficial. The distance function is a ruled surface with singularities, therefore using quadratic functions to approximate the distance function is not optimal. A richer library of local shape approximations could be generated in order to reconstruct accurately more complex sharp features. Finally, the MPU approach should be well suited to an out-of-core implementation due to the local nature of the weighting functions.

## Acknowledgments

The models are courtesy of the Digital Michelangelo Project 3D Model Repository (Michelangelo's David and Head of Michelangelo's David), the Stanford 3D Scanning Repository (Lucy, bunny, dragon, and Buddha), Cyberware (Igea, fandisk, and vase), Max-Planck-Institut für Informatik (Head of Max Planck), and the Imager Computer Graphics Laboratory of the University of British Columbia (knot).

## References

- ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *IEEE Visualization 2001*, 21–28.
- AMENTA, N., BERN, M., AND KAMVYSSELIS, M. 1998. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH 1998*, 415–421.
- BABUŠKA, I., AND MELENK, J. M. 1997. The partition of unity method. *International Journal of Numerical Methods in Engineering* 40, 727–758.
- BABUŠKA, CALOZ, G., AND OSBORN, J. E. 1994. Special finite element methods for a class of second order elliptic problems with rough coefficients. *SIAM J. Numerical Analysis* 31, 4, 745–981.
- BAJAJ, C. L., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of ACM SIGGRAPH 95*, 109–118.
- BEATSON, R. K., LIGHT, W. A., AND BILLINGS, S. 2000. Fast solution of the radial basis function interpolation equations: domain decomposition methods. *SIAM J. Sci. Comput.* 22, 5, 1717–1740.
- BERNARDINI, F., BAJAJ, C., CHEN, J., AND SCHIKORE, D. 1999. Automatic reconstruction of 3D CAD models from digital scans. *International Journal of Computational Geometry & Applications* 9, 4, 327–369.
- BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics* 1, 3 (July), 235–256.
- BLOOMENTHAL, J., BAJAJ, C., BLINN, J., CANI-GASCUEL, M. P., ROCKWOOD, A., WYVILL, B., AND WYVILL, G. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann.
- BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, 324–349.
- CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, 67–76.
- CURLESS, B. VripPack User's Guide. <http://graphics.stanford.edu/software/vrip/>.
- CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of ACM SIGGRAPH 1996*, 303–312.
- DINH, H. Q., SLABAUGH, G., AND TURK, G. 2001. Reconstructing surfaces using anisotropic basis functions. In *International Conference on Computer Vision (ICCV) 2001*, 606–613.
- DINH, H. Q., TURK, G., AND SLABAUGH, G. 2002. Reconstructing surfaces by volumetric regularization. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 24, 10 (October), 1358–1371.
- EDELSBRUNNER, H., AND MÜCKE, E. P. 1994. Three-dimensional alpha shapes. *ACM Transactions on Graphics* 13, 1 (January), 43–72.
- FLEISHMAN, S., COHEN-OR, D., ALEXA, M., AND SILVA, C. T. 2003. Progressive point set surfaces. *ACM Transactions on Graphics* 22, 4 (October).
- FRANKE, R., AND NIELSON, G. 1980. Smooth interpolation of large sets of scattered data. *International Journal of Numerical Methods in Engineering* 15, 1691–1704.
- FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, 249–254.
- GRIEBEL, M., AND SCHWEITZER, M. A. 2000. A Particle-Partition of Unity Method for the solution of Elliptic, Parabolic and Hyperbolic PDE. *SIAM J. Sci. Comp.* 22, 3, 853–890.
- GRIEBEL, M., AND SCHWEITZER, M. A. 2002. A Particle-Partition of Unity Method – Part III: A Multilevel Solver. *SIAM J. Sci. Comp.* 24, 2, 377–409.
- HART, J. C. 1996. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 527–545.
- HOPPE, H., DE ROSE, T., DUCHAMP, T., McDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized point. In *Proceedings of ACM SIGGRAPH 1992*, 71–78.
- HYPERFUN: F-REP LIBRARY. [http://cis.k.hosei.ac.jp/F-rep/HF\\_lib.html](http://cis.k.hosei.ac.jp/F-rep/HF_lib.html).
- ISKE, A., AND LEVESLEY, J. 2002. Multilevel scattered data approximation by adaptive domain decomposition. Tech. rep., University of Leicester, April.
- ISKE, A. 2001. Hierarchical scattered data filtering for multilevel interpolation schemes. In *Mathematical methods for curves and surfaces (Oslo, 2000)*. Vanderbilt Univ. Press, Nashville, TN, 211–221.
- JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. *ACM Transactions on Graphics* 21, 3 (July), 339–346. Proceedings of ACM SIGGRAPH 2002.
- KOBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH 2001*, 57–66.
- KOJEKINE, N., HAGIWARA, I., AND SAVCHENKO, V. 2003. Software tools using CSRBFs for processing scattered data. *Computers & Graphics* 27, 2 (April).
- LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The Digital Michelangelo Project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000*, 131–144.
- LIM, C., TURKIYAH, G. M., GANTER, M. A., AND STORTI, D. W. 1995. Implicit reconstruction of solids from cloud point sets. In *Proceedings of the third ACM symposium on Solid Modeling and Applications*, ACM Press, 393–402.
- MOORE, D., AND WARREN, J. 1991. Approximation of dense scattered data using algebraic surfaces. In *Proceedings of the 24th Hawaii International Conference on System Sciences*, IEEE Computer Society Press, Kauai, Hawaii, 681–690.
- MORSE, B. S., YOO, T. S., RHEINGANS, P., CHEN, D. T., AND SUBRAMANIAN, K. R. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modeling International 2001*, 89–98.
- MURAKI, S. 1991. Volumetric shape description of range data using “Bobby Model”. *Computer Graphics* 25, 4 (July), 227–235. Proceedings of ACM SIGGRAPH 1991.
- OHTAKE, Y., AND BELYAEV, A. G. 2002. Dual/primal mesh optimization for polygonized implicit surfaces. In *7th ACM Symposium on Solid Modeling and Applications*, 171–178.
- OHTAKE, Y., BELYAEV, A. G., AND SEIDEL, H.-P. 2003. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Shape Modeling International 2003*. Accepted.
- PASKO, A., AND SAVCHENKO, V. 1994. Blending operations for the functionally based constructive geometry. In *Set-theoretic Solid Modeling: Techniques and Applications, CSG 94 Conference Proceedings*, Information Geometers, 151–161.
- RENKA, R. J. 1988. Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software* 14, 2 (June), 139–148.
- RICCI, A. 1973. A constructive geometry for computer graphics. *The Computer Journal* 16, 2 (May), 157–160.
- SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNII, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum* 14, 4, 181–188.
- SCHABACK, R., AND WENDLAND, H. 2000. Adaptive greedy techniques for approximate solution of large RBF systems. *Numerical Algorithms* 24, 239–254.
- TAUBIN, G. 1991. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 13, 11, 1115–1138.
- TURK, G., AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *Proceedings of ACM SIGGRAPH 1994*, 311–318.
- TURK, G., AND O'BRIEN, J. 2002. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics* 21, 4 (October), 855–873.
- WARREN, J. 1992. Free-form blending: a technique for creating piecewise implicit surfaces. In *Topics in Surface Modeling*, H. Hagen, Ed. SIAM Press, Philadelphia, 473–483.
- WENDLAND, H. 2002. Fast evaluation of radial basis functions: Methods based on partition of unity. In *Approximation Theory X: Wavelets, Splines, and Applications*, Vanderbilt University Press, Nashville, L. Schumaker and J. Stöckler, Eds., 473–483.
- ZHAO, H., AND OSHER, S. 2002. Visualization, analysis and shape reconstruction of unorganized data sets. In *Geometric Level Set Methods in Imaging, Vision and Graphics*, Springer, S. Osher and N. Paragios, Eds.

# Interpolating and Approximating Implicit Surfaces from Polygon Soup



SIGGRAPH2004

Chen Shen  
James F. O'Brien  
Jonathan R. Shewchuk

University of California, Berkeley

## Introduction



SIGGRAPH2004

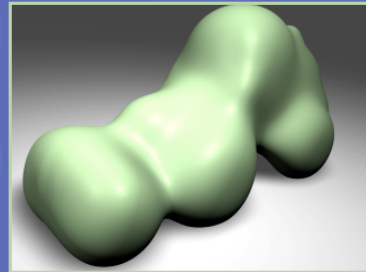
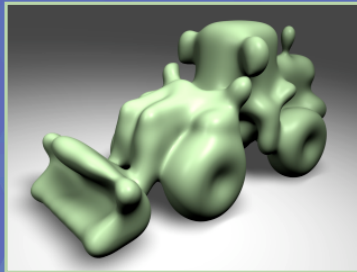
Polygonal Original



Implicit Surface



# Introduction



Approximating  
Surfaces



# Applications

- Repairing defective polygonal models
  - Holes, gaps, T-junctions, self-intersections, non-manifold structures
- Testing interior/exterior points
- Preprocessing for rapid prototyping machine
- Generating simulation envelopes



# Background

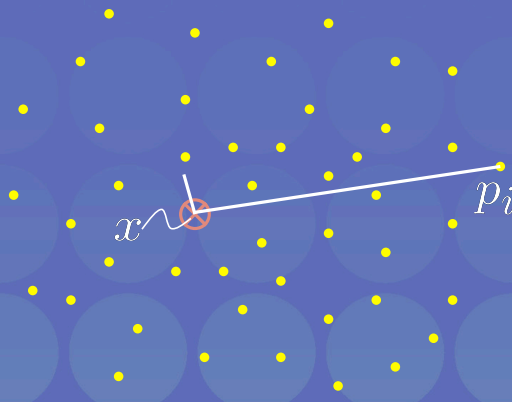
- Implicit Partition-of-Unity
  - Ohtake et al., 2003
- Moving Least Square projection method
  - Alexa et al., 2001, 2003, Fleishman et al., 2003, Amenta et al., 2004
- Other implicit techniques
- Other model fixing/smoothing methods

More detailed discussion in paper...



# Moving Least Square Basics

- Standard Least Square (“stationary”)



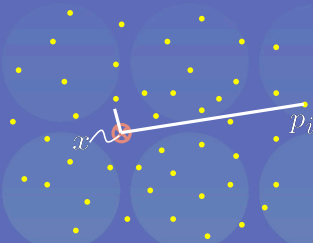


# Moving Least Square Basics



- Standard Least Square (“stationary”)

$$\begin{bmatrix} b^T(p_1) \\ \vdots \\ b^T(p_N) \end{bmatrix} c = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}$$



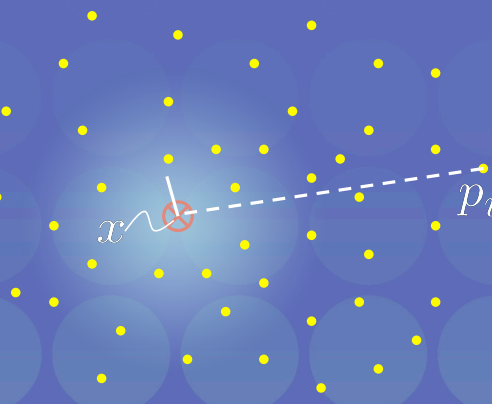
$$B^T B c = B^T \phi$$



# Moving Least Square Basics



- Moving Least Square

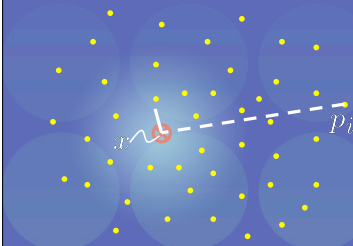


# Moving Least Square Basics



- Moving Least Square

$$\begin{bmatrix} w(x, p_1) \\ \vdots \\ w(x, p_N) \end{bmatrix} \begin{bmatrix} b^T(p_1) \\ \vdots \\ b^T(p_N) \end{bmatrix} c = \begin{bmatrix} w(x, p_1) \\ \vdots \\ w(x, p_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}$$



$$B^T (W(x))^2 B c(x) = B^T (W(x))^2 \phi$$

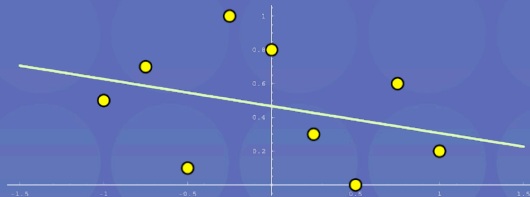
$$w(r) = \frac{1}{(r^2 + \epsilon^2)}$$



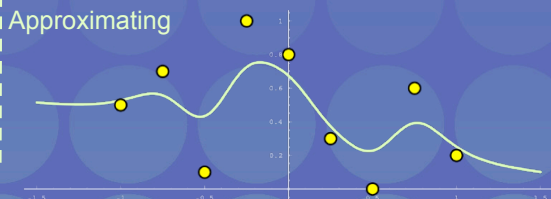
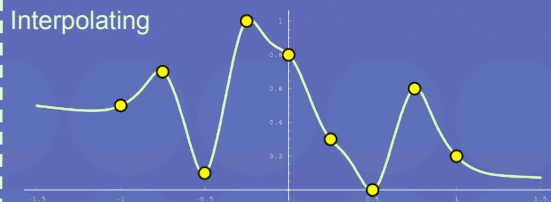
# Moving Least Square Basics



- Least Square



- Moving Least Square

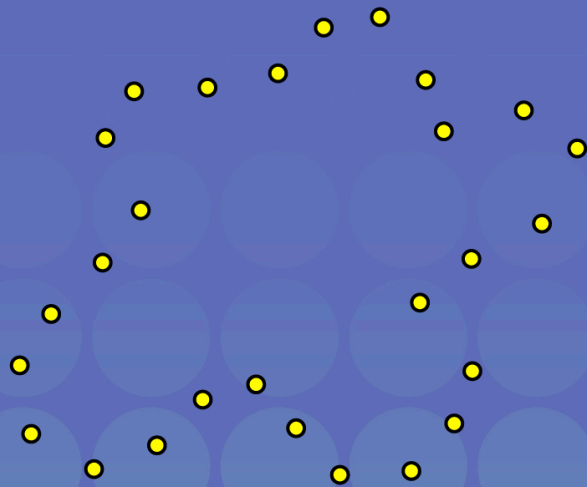
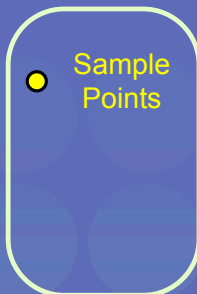


# Primary Innovations

- Implicit Moving Least Squares (IMLS)
- True normal constraints
  - No undesirable oscillatory behavior
- Integrated constraints over polygons
  - Avoids dimples and bumps
- Adjustment procedure
  - Tightly fit, completely enclosed
- Hierarchical fast evaluation

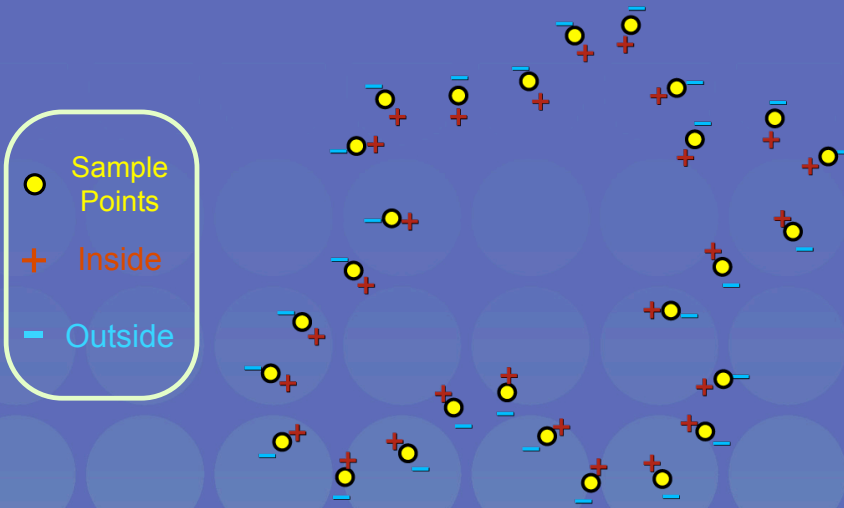


# Implicit MLS

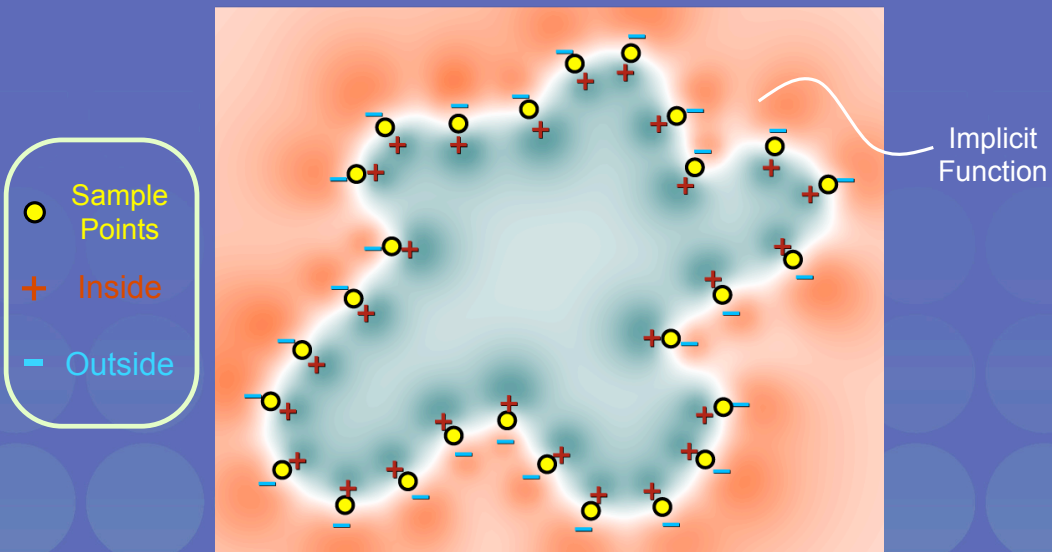




# Implicit MLS



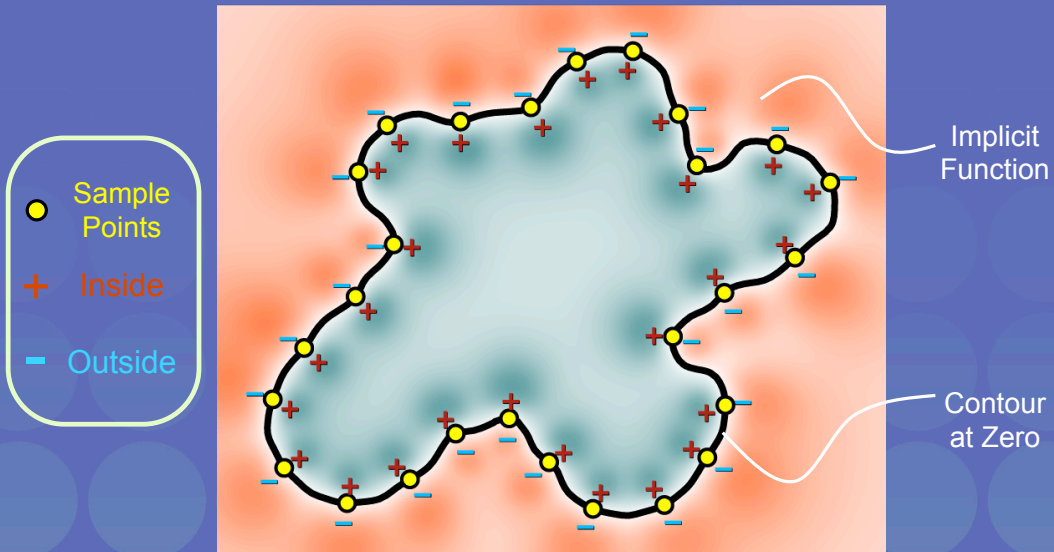
# Implicit MLS



# Implicit MLS



SIGGRAPH2004



# Normal Constraints

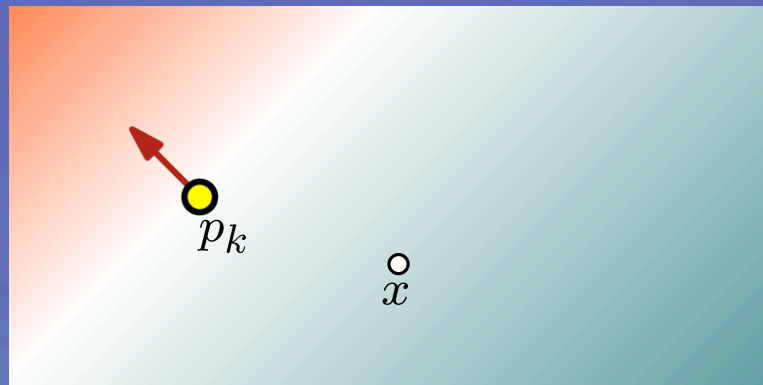


SIGGRAPH2004

- Problems with pseudo-normal constraints



# True Normal

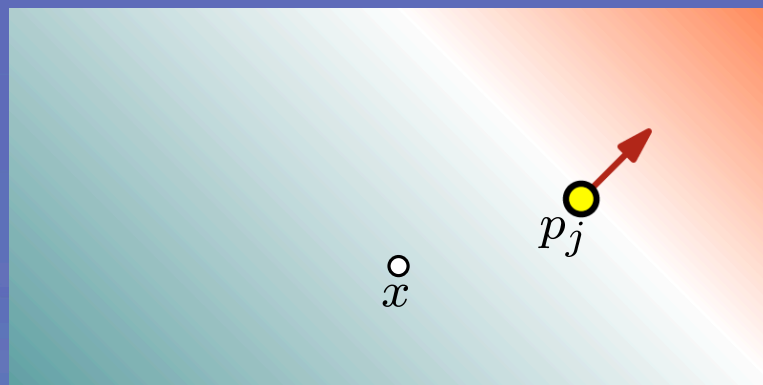


$$S_k(x) = \phi_k + (x - p_k)^T \hat{n}_k$$

$$= \psi_{0k} + \psi_{xk} x + \psi_{yk} y + \psi_{zk} z$$



# True Normal

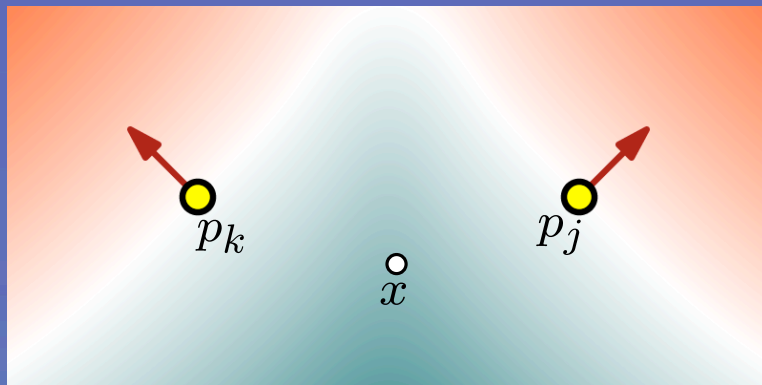


$$S_j(x) = \phi_k + (x - p_k)^T \hat{n}_k$$

$$= \psi_{0k} + \psi_{xk} x + \psi_{yk} y + \psi_{zk} z$$



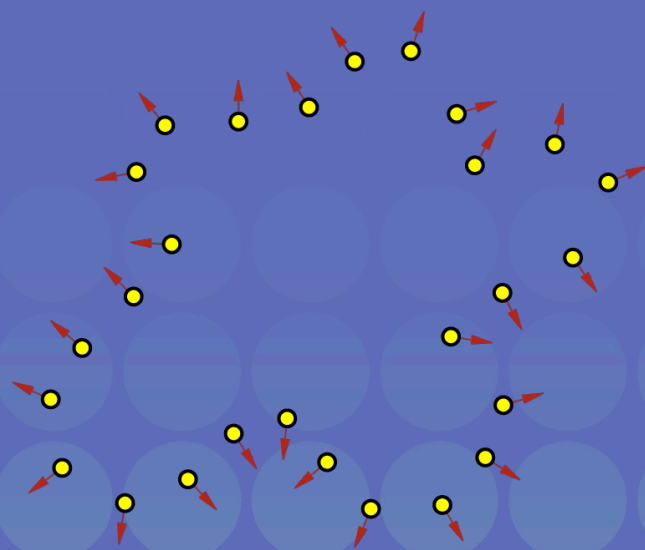
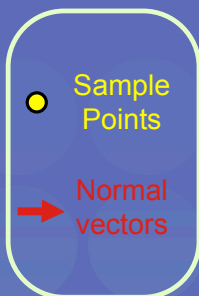
# True Normal



$$\begin{bmatrix} w(x, p_1) \\ \vdots \\ w(x, p_i) \end{bmatrix} c_1 = \begin{bmatrix} w(x, p_1) \\ \ddots \\ w(x, p_N) \end{bmatrix} \begin{bmatrix} S_1(x) \\ \vdots \\ S_N(x) \end{bmatrix}$$



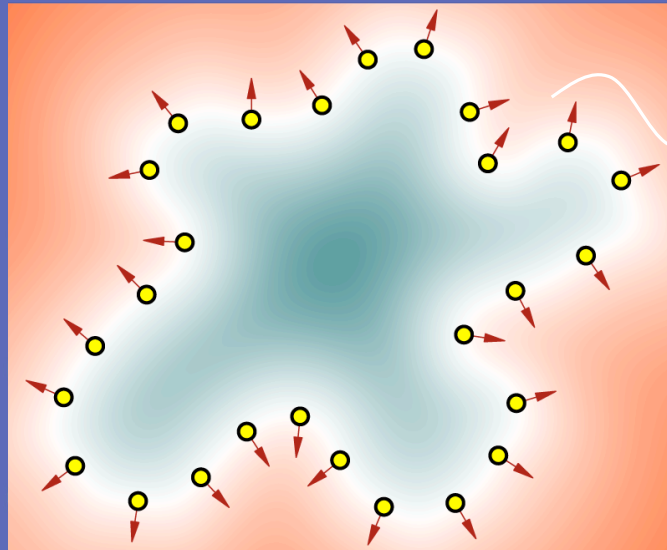
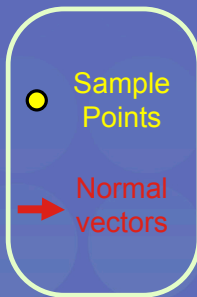
# Normal



# Normal



SIGGRAPH2004



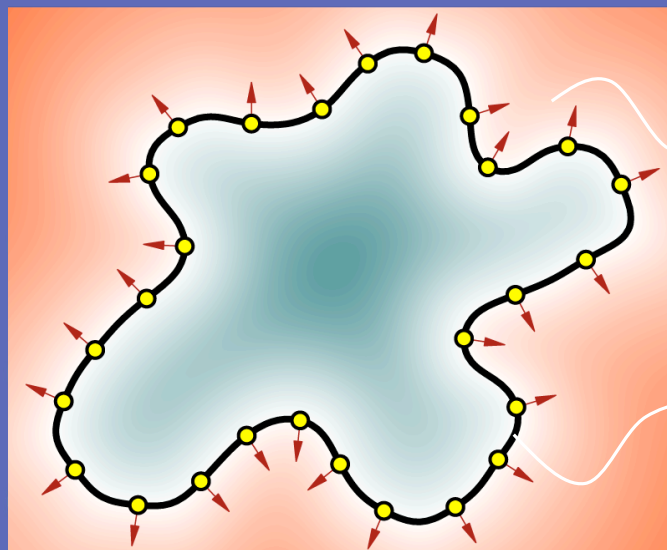
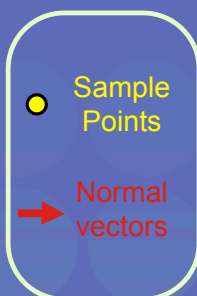
Implicit Function



# Normal



SIGGRAPH2004



Implicit Function

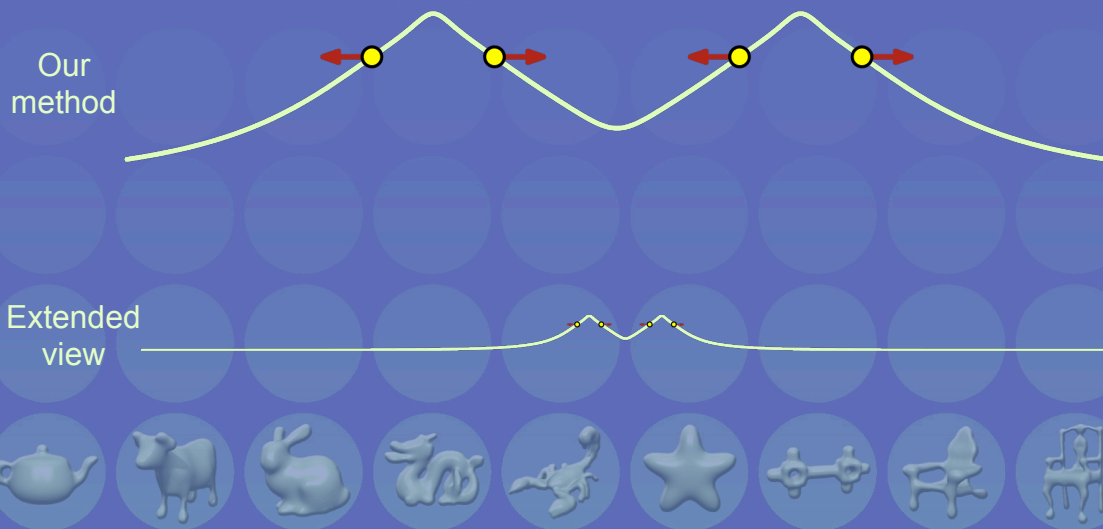
Contour at Zero



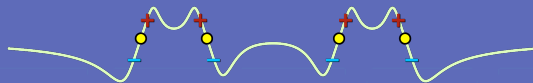


# Normal Constraints

- No undesirable oscillations



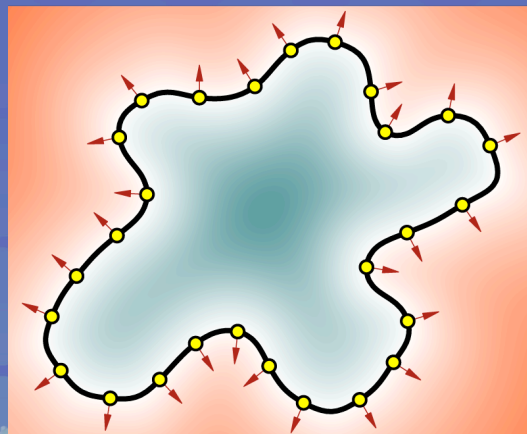
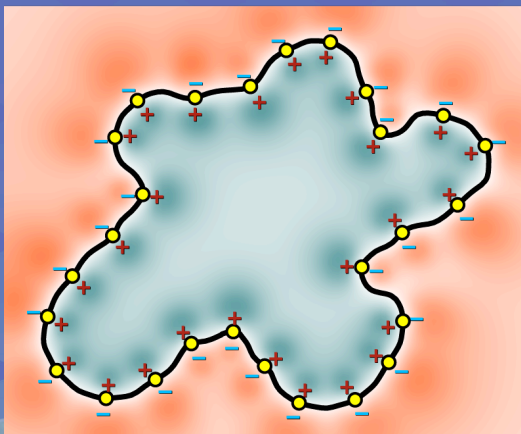
## Compare



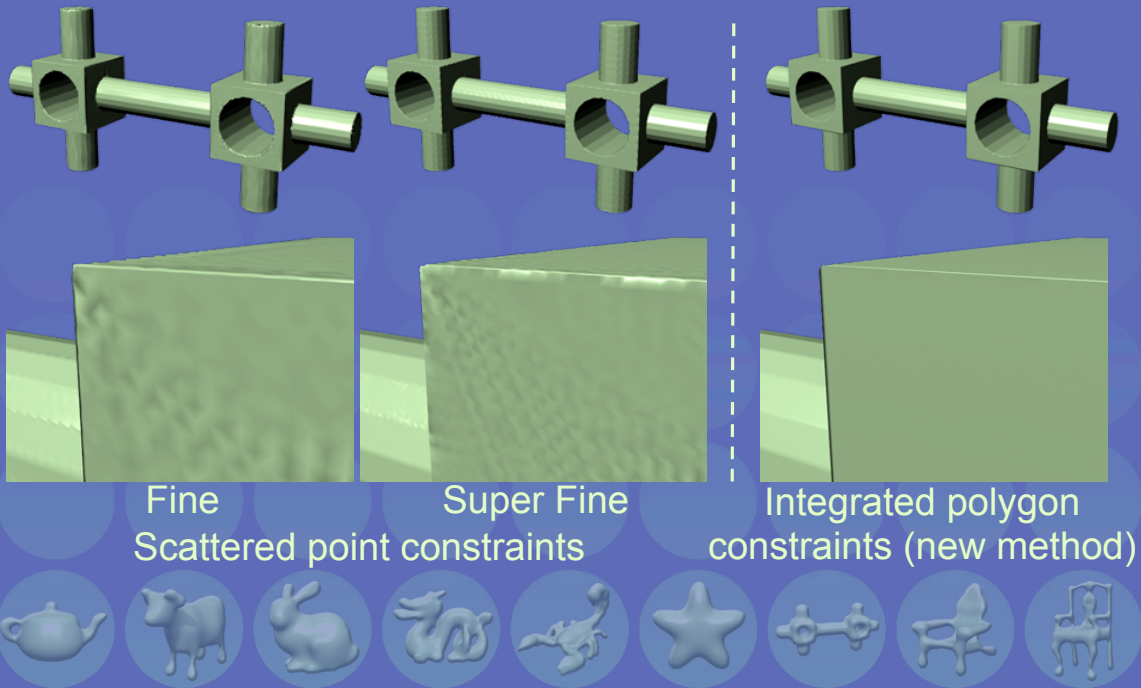
Point Constraints



Normal Constraints

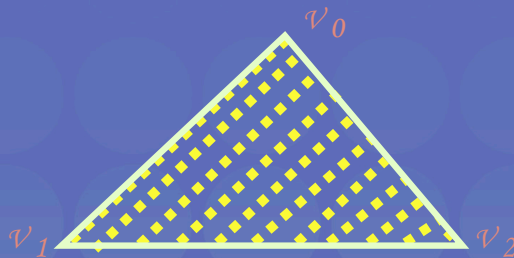


# Integrated over Polygons



# Integrated over Polygons

- Scattered points constraints

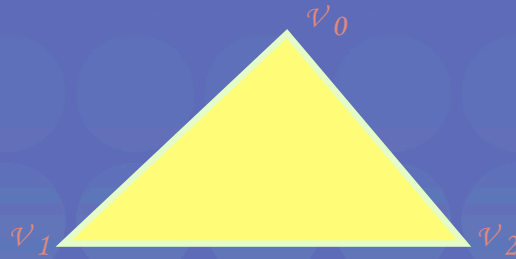


$$\left( \sum_{i=1}^N w^2(x, p_i) b(p_i) b^T(p_i) \right) c(x) = \sum_{i=1}^N w^2(x, p_i) b(p_i) \phi_i$$



# Integrated over Polygons

- Integral constraints

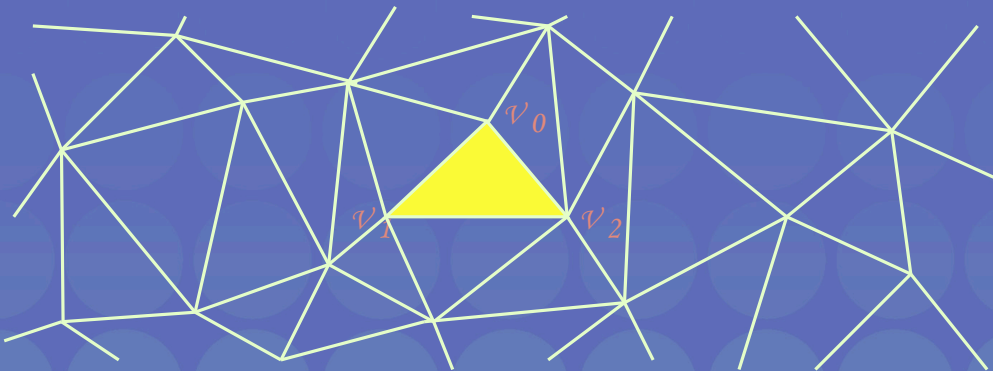


$$\left( \int_{\Omega_k} w^2(x, p) b(p) b^\top(p) dp \right) c(x) = \int_{\Omega_k} w^2(x, p) b(p) \phi_k dp$$



# Integrated over Polygons

- Polygons constraints



$$\sum_{k=1}^K \left( \int_{\Omega_k} w^2(x, p) b(p) b^\top(p) dp \right) c(x) = \sum_{k=1}^K \int_{\Omega_k} w^2(x, p) b(p) \phi_k dp$$





# Adjustment Procedure

- Naive approximation

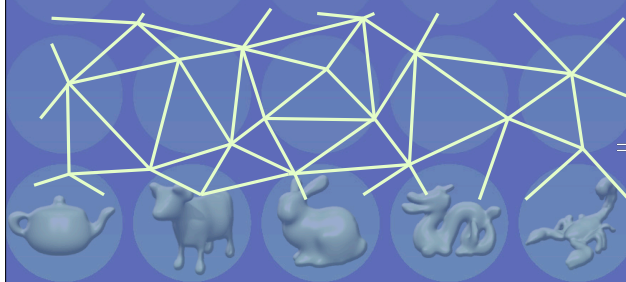


# Adjustment Procedure

- Iterative adjustment



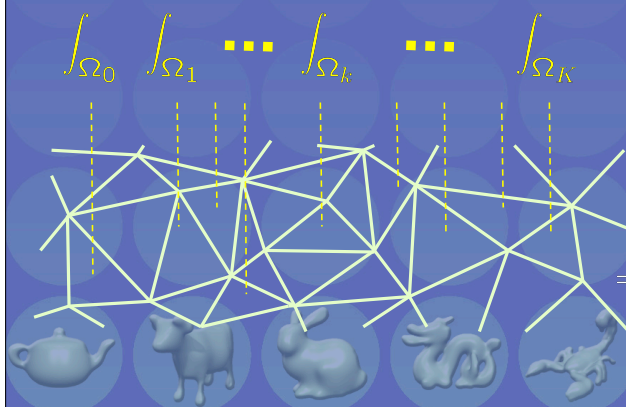
# Fast Evaluation



$$= \sum_{k=1}^K \left( \int_{\Omega_k} w^2(x, p) b(p) b^\top(p) \, dp \right) c(x)$$

$$= \sum_{k=1}^K \int_{\Omega_k} w^2(x, p) b(p) \phi_k \, dp$$

# Fast Evaluation

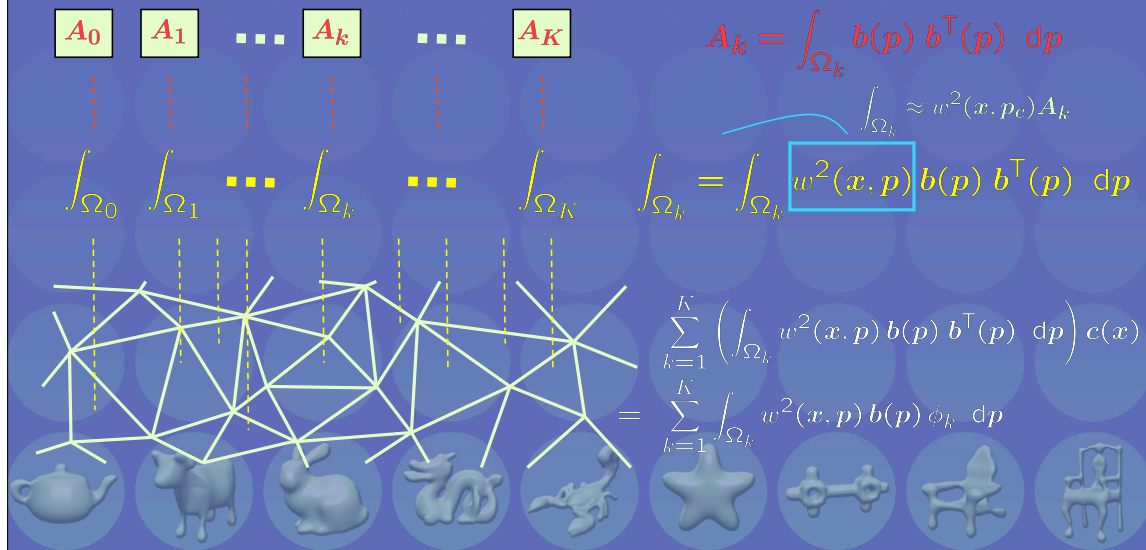


$$\int_{\Omega_0} \int_{\Omega_1} \dots \int_{\Omega_k} \dots \int_{\Omega_K} \int_{\Omega_k} = \int_{\Omega_k} w^2(x, p) b(p) b^\top(p) \, dp$$

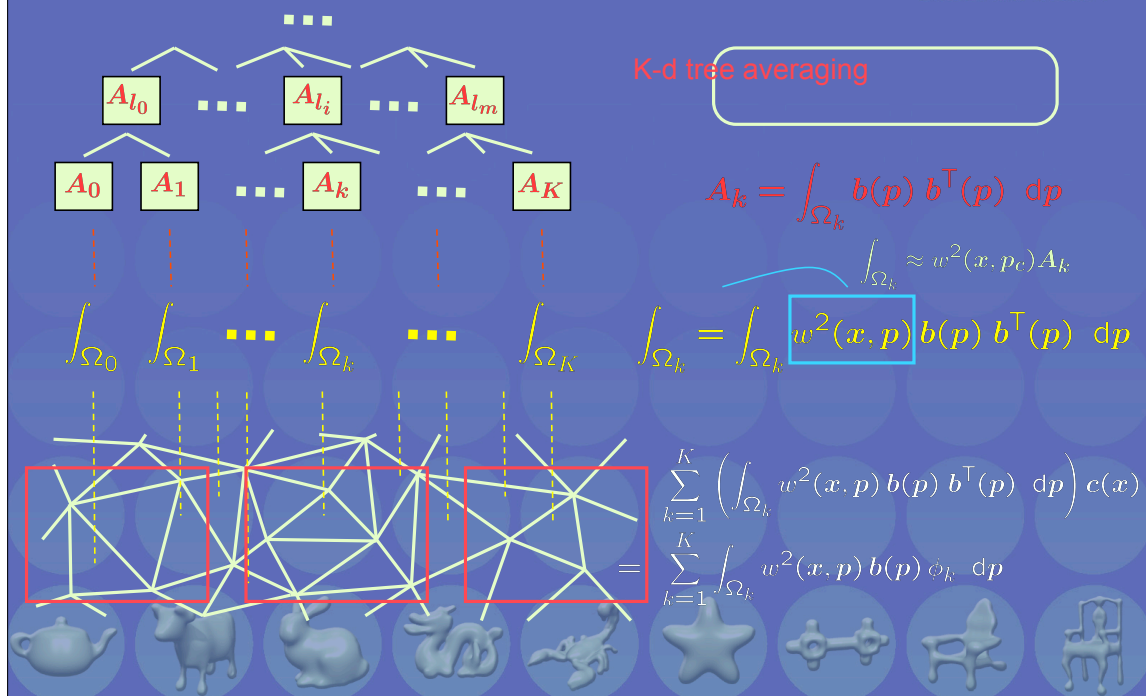
$$= \sum_{k=1}^K \left( \int_{\Omega_k} w^2(x, p) b(p) b^\top(p) \, dp \right) c(x)$$

$$= \sum_{k=1}^K \int_{\Omega_k} w^2(x, p) b(p) \phi_k \, dp$$

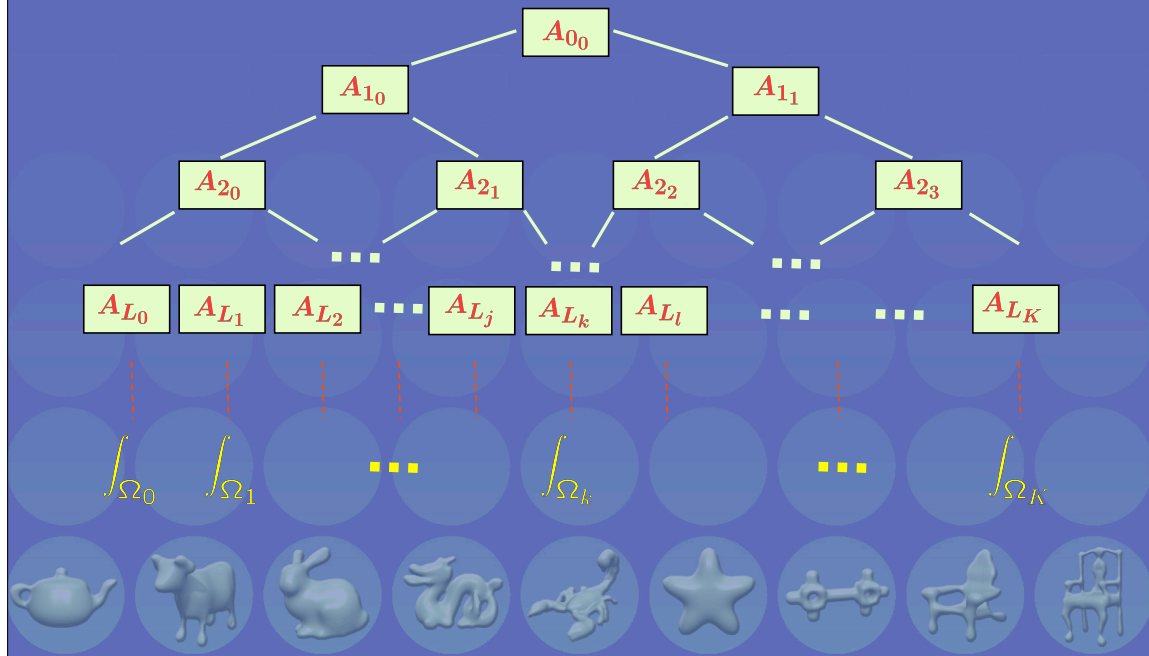
# Fast Evaluation



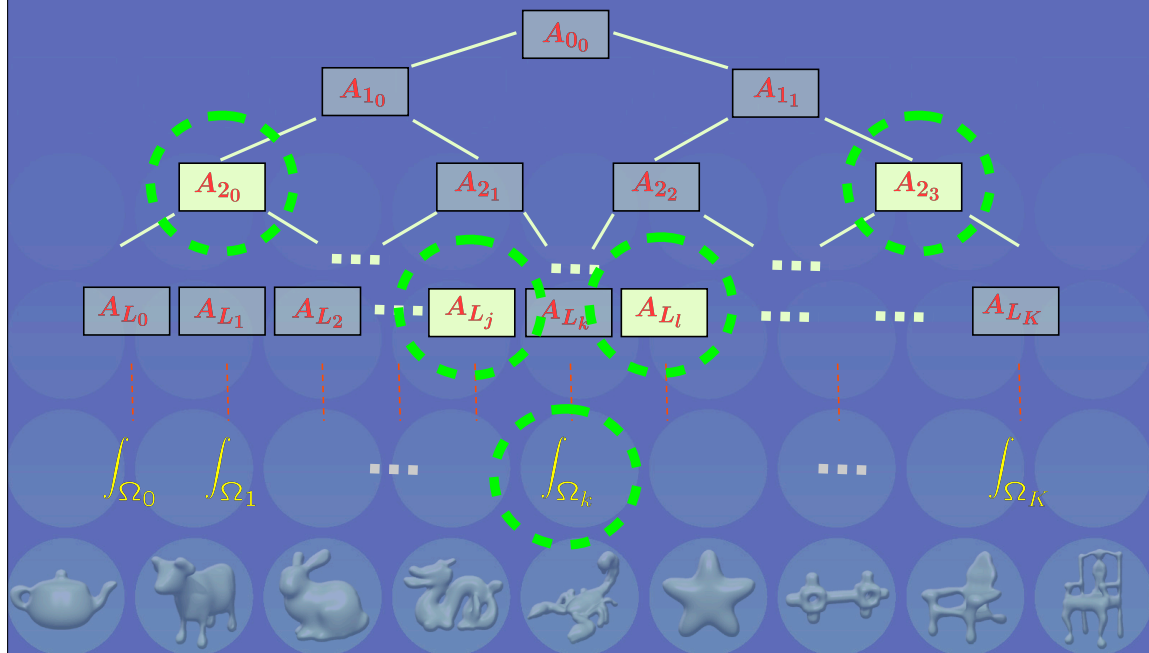
# Fast Evaluation



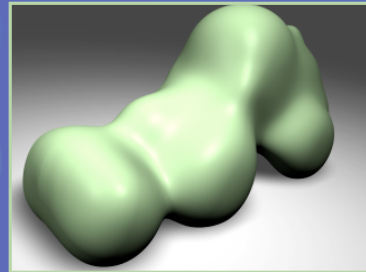
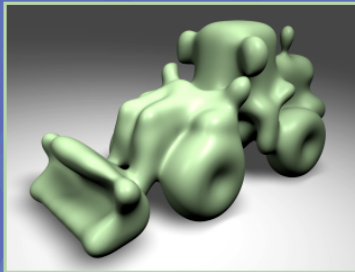
# Fast Evaluation



# Fast Evaluation



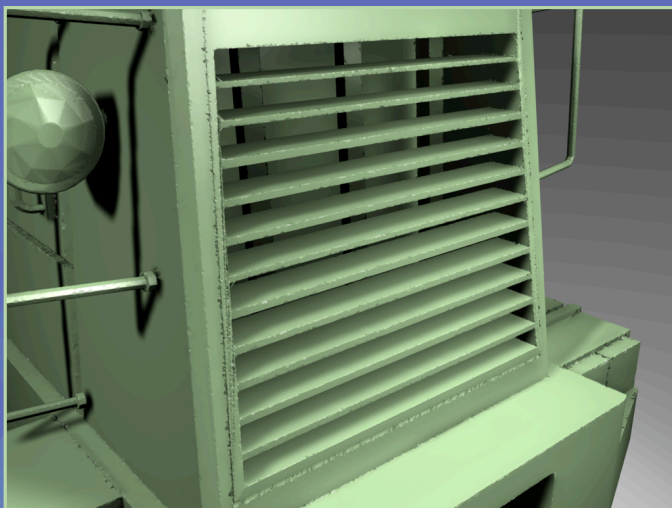
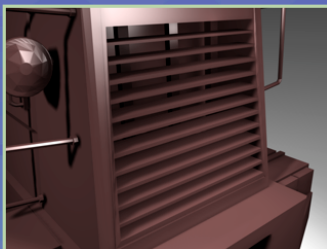
# Results



Approximating  
Surfaces



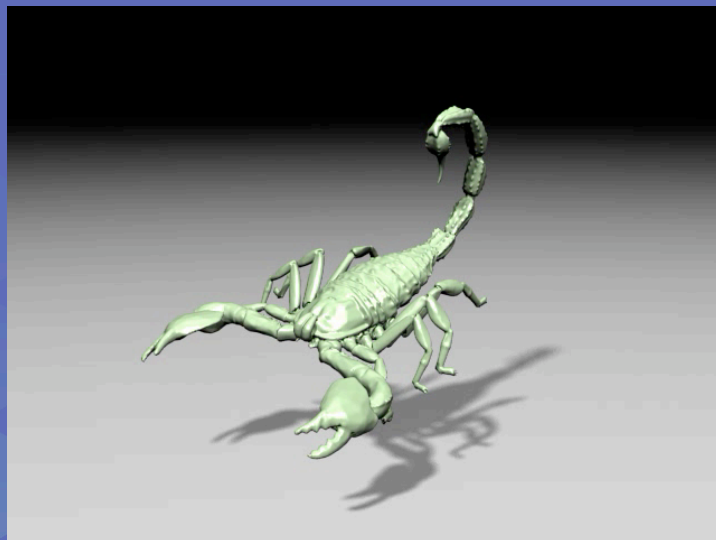
# Results







# Results



# Results



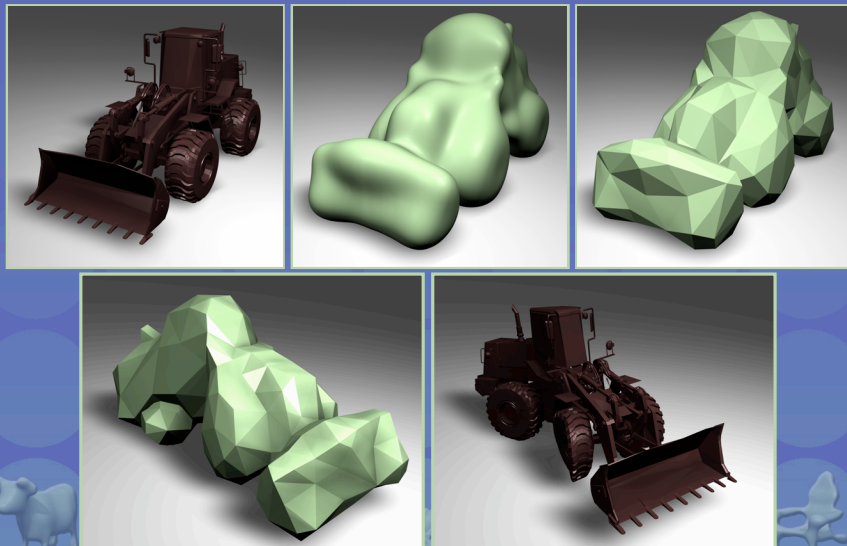
## Results

- Rapid prototyping



## Results

- Building simulation envelope





# Acknowledgements



- All members of Berkeley Graphics Group
- Ravi Kolluri for Polygonization
- Okan Arikan for Pixie
- Adam Bargteil for Simulation
- Prof. Carlo Sequin for FDM machine
- NSF CCR-0204377, MICRO 02-055, Pixar, Intel, Sony, Okawa Foundation and Alfred Sloan Foundation



# SIGGRAPH 2004





# Interpolating and Approximating Implicit Surfaces from Polygon Soup

Chen Shen

James F. O'Brien

Jonathan R. Shewchuk

University of California, Berkeley

## Abstract

This paper describes a method for building interpolating or approximating implicit surfaces from polygonal data. The user can choose to generate a surface that exactly interpolates the polygons, or a surface that approximates the input by smoothing away features smaller than some user-specified size. The implicit functions are represented using a moving least-squares formulation with constraints integrated over the polygons. The paper also presents an improved method for enforcing normal constraints and an iterative procedure for ensuring that the implicit surface tightly encloses the input vertices.

**Keywords:** Implicit surfaces, polygon soup, physically based animation, surface smoothing, topological simplification, simulation envelopes, point-based surfaces, surface representation, surface reconstruction.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; G.1.2 [Numerical Analysis]: Approximation—Approximation of surfaces and contours.

## 1 Introduction

Polygonal models occur ubiquitously in graphics applications. They are easy to render, easy to compute with, and a vast array of tools have been developed for creating and manipulating polygonal data. Unfortunately, polygonal data sets often contain problems, such as holes, gaps, T-junctions, self-intersections, and non-manifold structure, that make them unsuitable for many purposes other than rendering. Even when a polygonal data set does define a closed, manifold surface, other difficulties such as excessive detail or bad-aspect-ratio polygons, can preclude many uses. Data sets containing these problems are so common that the term “polygon soup” has evolved for describing arbitrary collections of polygons that carry no warranties concerning their structure.

This paper provides a tool that can transform arbitrary polygon data into a more useful form. We address this task with a method for generating implicit surfaces that can interpolate or approximate a set of polygons. The user controls how closely the surface approximates the input by selecting a minimum feature size. Geometric details or topological structures below this size tend to be smoothed away. Setting the minimum feature size to zero forces exact interpolation of the polygons. Additionally, if desired, we can cause an approximating surface to fit tightly around the input polygons

E-mail: {csh,job,jrs}@eecs.berkeley.edu

**From the ACM SIGGRAPH 2004 conference proceedings.**

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SIGGRAPH 2004, Los Angeles, CA

© Copyright ACM 2004

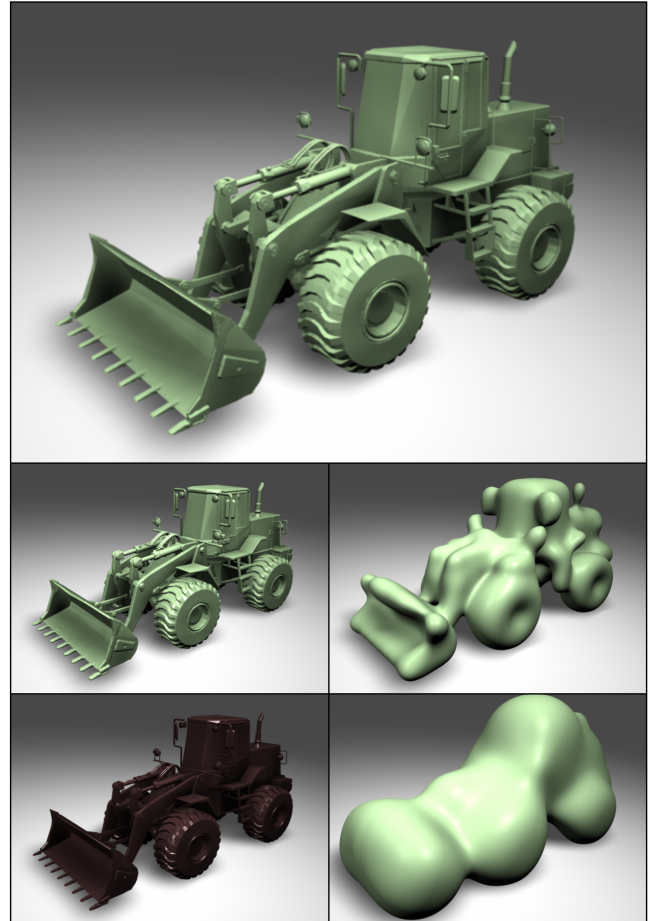


Figure 1: Interpolating and approximating surfaces (green) generated from polygonal original (brown).

while still ensuring that the input vertices are completely enclosed by the implicit surface. Figure 1 shows interpolating and approximating surfaces generated from a complex polygonal model with sharp edges and many small features.

An interpolating surface will exactly interpolate the input polygons, but it will also extend to fill gaps and holes so that the resulting surface will be “watertight.” This implicit function can then be used directly for a variety of applications, such as inside-outside tests, that are better suited to implicit representations. Alternatively, a clean polygonal model can be extracted and used for applications that require such clean polygonal input.

Approximating surfaces will naturally smooth out geometric features of the input data. Because we are using an implicit representation, topological structures of the input surfaces can also be smoothed away. This behavior makes the method suitable as part of a model simplification process when combined with an appropriate polygonization algorithm.

We can also force the approximating surface to stay “tight” around the original polygons while still smooth-

ing away details and ensuring that all the original vertices fall inside the approximating surface. This capacity allows us to generate a family of increasingly smooth approximations that eventually converge to a circumscribing ellipsoid. Among other uses, these simplified shapes can be used for easily generating efficient simulation envelopes.

Our algorithm makes use of a scattered-data interpolation method known as *moving least-squares*, commonly abbreviated MLS. The function defining our implicit surfaces is specified by the moving least-squares solution to a set of constraints that would force the function to a given value over the surface region of each polygon, and that would over the same region also force the function's upward gradient to match the polygon's outward normal. Neither condition is specified by simple point constraints: integrated constraints are used over each polygon, and normals constraints directly affect the function's gradient. The degree of approximation is controlled by simply adjusting the least-squares weighting function, but the tightness of the surface and the requirement that the input vertices fall inside the implicit surface both depend on an iterative procedure for adjusting the constraint values over each polygon.

The moving least-squares method has been used by other researchers to define a surface as the fixed-point of an iterative parametric fit procedure—for example, see [Alexa et al., 2001]. Other than using the same general mathematical tool, that approach and this one are unrelated. Unfortunately, those surfaces are often referred to simply as *MLS Surfaces* which may cause some confusion with the method described here. We suggest that the term *implicit moving least-squares surface*, or *IMLS Surface* be used to describe our method.

Our approach is, however, closely related to implicit methods based on partition-of-unity interpolants. (For example see [Ohtake et al., 2003a].) Partition-of-unity and moving least-squares interpolants use different notation, but they are fundamentally alike. One key difference between our formulation and prior ones is that our integrated constraints differ significantly from collections of point constraints. We also use improved normal and approximation procedures, which are applicable to point constraints as well as to our integrated constraints.

Our algorithm has five primary components:

- A scattered data interpolation scheme that, in addition to simple point constraints, allows integrated constraints over polygons.
- A method for enforcing true normal constraints that does not produce undesirable oscillatory behavior.
- An adjustment procedure that causes the implicit surface to fit tightly around the input polygons while still ensuring that the input vertices are completely enclosed by the implicit surface.
- A hierarchical fast evaluation scheme that makes the method practical for large data sets.
- Optional preprocessing to remove unwanted geometry and enforce consistency among the input normals.

## 2 Background

The work most closely related to ours appears in [Ohtake et al., 2003a]. They use a partition-of-unity method to build a function whose zero-set passes through, or near, a set of input points. Using a procedure originally proposed by [Turk and O'Brien, 1999], they place zero-constraints at each input point, and they also place a pair of additional non-zero point constraints offset in the inward and outward normal directions. To keep the method feasible for large data sets, they use a fast hierarchical evaluation scheme. The partition-of-unity formulation they use and the moving least-squares formulation that we start with are essentially

identical: they both belong to a family of meshless interpolation methods that also includes the element-free Galerkin method and smoothed particle hydrodynamics. We refer the reader to [Belytschko et al., 1996] for a discussion of the relationships between these different formulations. The two most significant differences between our work and [Ohtake et al., 2003a] are that we use integrated polygon constraints, and that we use a significantly improved method for enforcing normal constraints. We also describe a different hierarchical evaluation scheme and an iterative method for generating useful approximating surfaces.

Moving least-squares interpolation is also a part of the non-linear projection method used in [Alexa et al., 2001], [Alexa et al., 2003], and [Fleishman et al., 2003]. This projection method defines a surface as a function of a set of points, but the moving least-squares fit is used as part of a non-linear projection that differs substantially from the implicit-surface based method described here.

The technique of defining a surface implicitly using a function constrained to match a set of input points is fairly widespread. In [Savchenko et al., 1995], [Turk and O'Brien, 1999], [Carr et al., 2001], and [Turk and O'Brien, 2002] the function is represented using globally supported radial splines. This class of functions has the nice property that one can make definite statements about a solution's global behavior. These radial splines have also been used to match polygon data by [Yngve and Turk, 2002]. While they were able to achieve results that roughly matched the input polygons, the resulting implicit surfaces still deviated substantially from the input. Different, locally supported functions were used in both [Muraki, 1991], [Morse et al., 2001], and [Ohtake et al., 2003b] for fitting an implicit surface to clouds of point data. In addition to representing function as sums of continuous basis functions, [Museth et al., 2002] and [Zhao et al., 2001] have used level-set methods for fitting surfaces to point clouds. Other function representations include signed-distance functions [Cohen-Or et al., 1998], and medial axes [Bittar et al., 1995]. The text, [Bloomenthal, 1997], also describes several other methods for representing implicit surfaces.

Some of the applications that can be addressed with our method have also been addressed with other methods. An enormous amount of work has been done on smoothing explicit representations of polygonal models, two early examples of which include [Taubin, 1995] and [Desbrun et al., 1999]. Work in that subarea is now quite advanced and methods are available that can preserve sharp features while still smoothing away noise. (For a single recent example, see [Jones et al., 2003].) We can also generate envelopes around input objects and similar ideas have been explored in [Cohen et al., 1996] and [Keren and Gotsman, 1998]. The problem of rectifying polygonal models has been investigated in [Nooruddin and Turk, 2003]. In [Nooruddin and Turk, 2000] the same researchers also looked at methods for removing unwanted interior structure from a polygon model.

## 3 Methods

The primary tool we work with is a scattered data interpolation method known as moving least-squares. With this method we can create an implicit surface that either interpolates or approximates a given polygonal surface. In this section, we describe how we set up and apply constraints that allow us to generate and control the behavior of the implicit surface.

For the sake of clear exposition, we will start by describing a moving least-squares method for defining implicit surfaces using simple point constraints. We will then describe how that method can be extended to include integrated constraints defined over polygonal regions. Once we specify the

framework we use for defining our functions, we will describe how we enforce normal constraints, adjust the tightness of the surface around the input, and preprocess the data to avoid unwanted internal structures.

During our discussion of the implicit moving-least squares formulation, we keep the description of basis and weighting functions general. However, although our implementation supports a wide range of function choices, we have found that simple weighting functions and constant basis functions are computationally inexpensive, yet they produce results just as good as more expensive choices. For other problems, different choices of weighting and basis functions may be useful.

### 3.1 Value Constraints at Points

Assume that we have  $N$  points located at positions  $\mathbf{p}_i$ ,  $i \in [1 \dots N]$ , and we would like to build a function,  $f(\mathbf{x})$ , that approximates the values  $\phi_i$  at those points. For a standard least-squares fit we would solve

$$\begin{bmatrix} \mathbf{b}^\top(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}^\top(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}, \quad (1)$$

where  $\mathbf{b}(\mathbf{x})$  is the vector of basis functions we use for the fit, and  $\mathbf{c}$  is the unknown vector of coefficients. Unless this system is under-constrained, it can be resolved efficiently using the method of normal equations and solving an  $M \times M$  linear system, where  $M$  is the number of basis functions (*i.e.*, the lengths of  $\mathbf{b}$  and  $\mathbf{c}$ ). For example, if we wished to fit a plane we would choose  $\mathbf{b}(\mathbf{x}) = [1, x, y, z]$ , or simply  $\mathbf{b}(\mathbf{x}) = [1]$  if we just wished to fit a constant. The resulting function is

$$f(\mathbf{x}) = \mathbf{b}^\top(\mathbf{x}) \mathbf{c}. \quad (2)$$

For the moving least-squares formulation, we allow the fit to change depending on where we evaluate the function so that  $\mathbf{c}$  varies with  $\mathbf{x}$ . We do so by weighting each row of Equation (1) by  $w(\|\mathbf{x} - \mathbf{p}_i\|)$ , where  $w(r)$  is some distance weighting function, which gives us

$$\begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) \\ \vdots \\ w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \mathbf{b}^\top(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}^\top(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) \\ \vdots \\ w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \quad (3)$$

where  $w(\mathbf{x}, \mathbf{p}_i) = w(\|\mathbf{x} - \mathbf{p}_i\|)$ .

By selecting an appropriate weight function, a variety of interpolating or approximating behaviors can be achieved, even with low-order basis functions. In general, a weight function that approaches  $+\infty$  at zero will cause interpolation. We use the weight function

$$w(r) = \frac{1}{(r^2 + \epsilon^2)}. \quad (4)$$

The parameter  $\epsilon$  allows a degree of control over the function's behavior which we discuss later.

Giving matrices names and explicitly noting their dependence on  $\mathbf{x}$ , Equation (3) becomes

$$\mathbf{W}(\mathbf{x}) \mathbf{B} \mathbf{c}(\mathbf{x}) = \mathbf{W}(\mathbf{x}) \boldsymbol{\phi}. \quad (5)$$

The resulting normal equations are

$$\mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \mathbf{B} \mathbf{c}(\mathbf{x}) = \mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \boldsymbol{\phi} \quad (6)$$

and we can evaluate the fit function's value using

$$f(\mathbf{x}) = \mathbf{b}^\top(\mathbf{x}) \mathbf{H}^{-1} \mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \boldsymbol{\phi}, \quad (7)$$

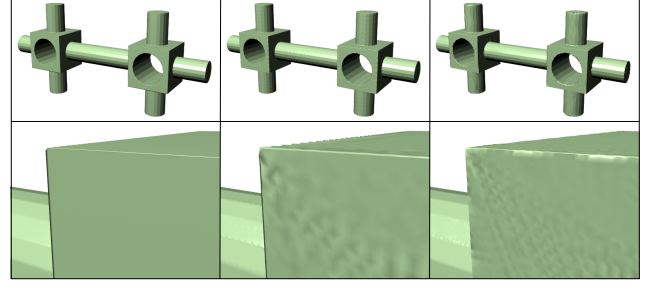


Figure 2: The column on the left shows the results generated using integrated polygonal constraints. The middle and right columns show the results generated with different densities of scattered point constraints.

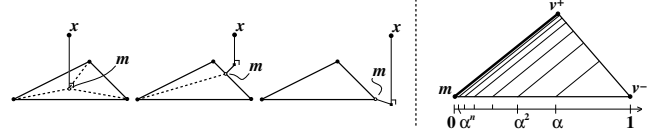


Figure 3: The quadrature scheme used over a triangle.

where

$$\mathbf{H} = \mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \mathbf{B}. \quad (8)$$

The derivatives with respect to  $\mathbf{x}$  of the fit function can be evaluated using

$$\begin{aligned} f'(\mathbf{x}) &= (\mathbf{b}^\top)'(\mathbf{x}) \mathbf{H}^{-1} \mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \boldsymbol{\phi} - \\ &\quad \mathbf{b}^\top(\mathbf{x}) \mathbf{H}^{-1} \mathbf{H}' \mathbf{H}^{-1} \mathbf{B}^\top (\mathbf{W}(\mathbf{x}))^2 \boldsymbol{\phi} + \\ &\quad \mathbf{b}^\top(\mathbf{x}) \mathbf{H}^{-1} \mathbf{B}^\top ((\mathbf{W}(\mathbf{x}))^2)' \boldsymbol{\phi}, \end{aligned} \quad (9)$$

where

$$\mathbf{H}' = \mathbf{B}^\top ((\mathbf{W}(\mathbf{x}))^2)' \mathbf{B}, \quad (10)$$

and the derivative of  $(\mathbf{W}(\mathbf{x}))^2$  is obtained by simply taking the derivative of the squared weighting function along the matrix's diagonal.

### 3.2 Value Constraints Integrated over Polygons

Although the formulation in the previous section works well for point constraints, the input data we are concerned with consists of polygons, and for each of these polygons we want to constrain the fit function over its entire surface. If we were not interested in interpolating the polygons, we could approximate the desired effect with point constraints scattered over the surface of each polygon. Aside from potentially requiring a very large number of points, scattered point constraints work reasonably well for approximating surfaces. However, interpolating surfaces and surfaces that approximate closely show undesirable bumps and dimples corresponding to the point locations. (See Figure 2.) In particular, bumps and dimples occur unless  $\epsilon$  is substantially larger than the spacing between points.

To achieve good results, what we would like to do is to scatter an infinite number of points continuously across the surface of each polygon. Notice that Equation (6) can be rewritten as an explicit summation over a set of point constraints,

$$\left( \sum_{i=1}^N w^2(\mathbf{x}, \mathbf{p}_i) \mathbf{b}(\mathbf{p}_i) \mathbf{b}^\top(\mathbf{p}_i) \right) \mathbf{c}(\mathbf{x}) = \sum_{i=1}^N w^2(\mathbf{x}, \mathbf{p}_i) \mathbf{b}(\mathbf{p}_i) \phi_i \quad (11)$$

In this form it becomes clear how we can apply constraints continuously over each polygon's surface.

For a data set of  $K$  polygons, let  $\Omega_k$ ,  $k \in [1 \dots K]$ , be the  $k$ th input polygon. The parenthesized term of Equation (11)

and the term on the right are replaced by integrals over the polygons and we have

$$\left( \sum_{k=1}^K A_k \right) c(\mathbf{x}) = \sum_{k=1}^K \mathbf{a}_k \quad (12)$$

where  $A_k$  and  $\mathbf{a}_k$  are defined by

$$A_k = \int_{\Omega_k} w^2(\mathbf{x}, \mathbf{p}) \mathbf{b}(\mathbf{p}) \mathbf{b}^\top(\mathbf{p}) d\mathbf{p} \quad , \quad (13)$$

$$\mathbf{a}_k = \int_{\Omega_k} w^2(\mathbf{x}, \mathbf{p}) \mathbf{b}(\mathbf{p}) \phi_k d\mathbf{p} \quad , \quad (14)$$

$\mathbf{p}$  is the integration variable ranging over the polygon, and  $\phi_k$  is the constraint value. We can choose  $\phi_k$  to be constant, or we can choose  $\phi_k$  to vary polynomially over each polygon. For later use, it is convenient to define terms with the weighting function omitted:

$$\tilde{A}_k = \int_{\Omega_k} \mathbf{b}(\mathbf{p}) \mathbf{b}^\top(\mathbf{p}) d\mathbf{p} \quad , \quad (15)$$

$$\tilde{\mathbf{a}}_k = \int_{\Omega_k} \mathbf{b}(\mathbf{p}) \phi_k d\mathbf{p} \quad . \quad (16)$$

The integrals will be infinite when  $\epsilon = 0$  and the evaluation point  $\mathbf{x}$  lies precisely on a polygon. In this case,  $f(\mathbf{x})$  has a removable singularity at  $\mathbf{x}$ ; we can skip the least-squares step and simply set  $f(\mathbf{x})$  to the value  $\phi_k$  dictated by the polygon. It is possible that two polygons intersect at a point where their constraints disagree, in which case  $f$  has an essential singularity at that point. Evaluating at or near such points in a numerically stable fashion is difficult. However, we can sidestep the issue by setting  $\epsilon$  to an extremely small number, far below the smallest feature size relevant to a given application.

Computing these integrals is conceptually straightforward. Each entry of the matrix  $\mathbf{b} \mathbf{b}^\top$  and the vector  $\mathbf{b}$  is a polynomial in  $\mathbf{p}$ , the weight function we have chosen is a rational polynomial in  $\mathbf{p}$ , and each of the components of the matrices can, of course, be computed independently. For a one-dimensional integral (*i.e.*, constraints over edges) the integrals have closed form solutions. (See Appendix A.) Unfortunately, we have not been able to find closed-form solutions of the two-dimensional integrals.

The obvious solution to this problem would simply approximate the integrals using a standard quadrature method. Unfortunately, this solution performs poorly for the same reason that scattering point constraints does: unless the distance between quadrature points is significantly less than  $\epsilon$  the resulting surface will have dimples and bumps. The culprit responsible for this behavior is the weighting function. Its singularity, or near singularity, at zero, causes severe problems for standard quadrature schemes. These difficulties extend to Monte-Carlo schemes, which explains the problems encountered with scattered points. The method we use is aware of the singular nature of the weighting function and it accounts for that contribution without underweighting the contribution from the rest of the triangle.

Let  $\mathbf{m}$  be the point in  $\Omega_k$  that is closest to the evaluation point,  $\mathbf{x}$ . (See Figure 3.) If this point is on the interior of  $\Omega_k$ , we split the triangle into three triangles each of which has  $\mathbf{m}$  as one of its vertices. If the point lies on an edge, the triangle is split into two triangles. If the point lies on an existing vertex, the triangle is not split. The integral over the original triangle is the sum of integrals over each of these sub-triangles. Each sub-triangle has  $\mathbf{m}$  as one of its vertices,

and the other two vertices are denoted  $\mathbf{v}^+$  and  $\mathbf{v}^-$  such that  $w(\mathbf{x}, \mathbf{v}^+) \geq w(\mathbf{x}, \mathbf{v}^-)$ .

To compute the sub-triangle area integral we separate it into two successive one-dimensional integrals as shown in Figure 3. The outer one integrates along the edge from  $\mathbf{m}$  to  $\mathbf{v}^-$  using a special numerical quadrature rule. The inner one integrates along the barycentric iso-lines that are parallel to the edge from  $\mathbf{m}$  to  $\mathbf{v}^+$ , using the one-dimensional analytical solution.

The outer, numerical integration uses the Newton-Cotes trapezoidal rule with irregularly spaced samples. If the edge from  $\mathbf{m}$  to  $\mathbf{v}^-$  is parameterized from zero to one with zero corresponding to  $\mathbf{m}$ , the samples occur at  $0, \alpha^n, \dots, \alpha^1, \alpha^0$ . We arbitrarily use  $\alpha = 2/3$ , and  $n$  is proportional to the logarithm of the edge length. The integral should be appropriately scaled by the sub-triangle area. This scheme captures the behavior near the potentially singular location,  $\mathbf{m}$ , without neglecting the rest of the triangle.

### 3.3 Normal Constraints

The two previous sections describe how we can implement constraints on the value of the moving least-squares function at discrete points and over polygonal patches. However, if we attempt to define a surface by only requiring it to take a given value on its surface, we will not obtain useful results. Previous researchers, for example [Ohtake et al., 2003a], have implemented pseudo-normal constraints with a technique originally suggested by [Turk and O'Brien, 1999]. This technique places a zero constraint at a point on the surface, a positive constraint offset slightly outside the surface, and a negative one slightly inside.

Unfortunately, this approach does not work as well as one might like. The additional constraints influence the function's gradient only crudely, and they can cause undesirable oscillatory behavior as the evaluation point moves away from the surface. This behavior is illustrated in the lower half of Figure 4. It occurs because when the distance between the evaluation point and the surface point is much larger than the offset distance, the inside and outside constraints effectively cancel each other out. Even if only outside (or only inside) constraints are used, they will still effectively merge to a single average valued constraint far away. Heuristics, such as those described by [Ohtake et al., 2003a], can suppress some of the spurious behavior, but the value of the function far from the surface will not be useful. Furthermore, these quasi-normal constraints cause severe problems when used with the approximation procedure described in the next section.

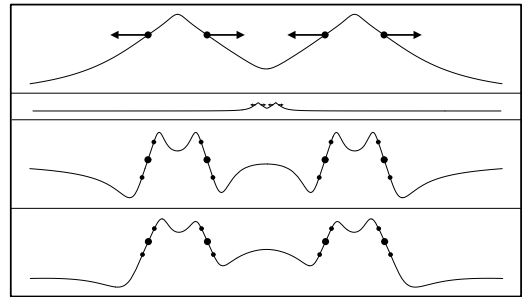


Figure 4: A one-dimensional example showing the height field generated from four position and normal constraints. The first (top) image shows the result with our method, and the arrows indicate the outward normal directions. The second shows an expanded view demonstrating far-field behavior. The third and fourth images show the results generated by pseudo-normal constraints with linear and quadratic basis functions. The small dots indicate the placement of the inside and outside pseudo-normal constraints.



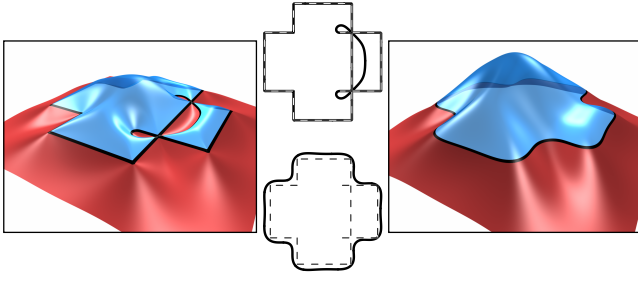


Figure 5: A two-dimensional example comparing interpolating and approximating results. The center images show input constraints as dotted lines and the contour as a solid line. The outer images show the resulting function as a height-field.

One of our key innovations is to impose normal constraints by forcing the interpolating function to behave like a prescribed function (in the neighborhood of a polygon), as opposed to a prescribed constant value. In other words, instead of using the moving least-squares method to blend between *constant values* associated with each polygon (or point), we blend between *functions* associated with them. This method exhibits little undesirable oscillation.

If  $\hat{n}_k$  is the normal associated with polygon  $\Omega_k$ , we define the function  $S_k(\mathbf{x})$  that describes how that polygon wants the interpolant to behave as

$$S_k(\mathbf{x}) = \phi_k + (\mathbf{x} - \mathbf{q}_k)^\top \hat{n}_k \quad (17)$$

$$= \psi_{0k} + \psi_{xk}x + \psi_{yk}y + \psi_{zk}z, \quad (18)$$

where  $\mathbf{q}_k$  is an arbitrary point on the polygon  $\Omega_k$ , and  $\psi_{0k}$ ,  $\psi_{xk}$ ,  $\psi_{yk}$ , and  $\psi_{zk}$  are resulting polynomial coefficients. Interpolating between these functions reduces to simply interpolating the  $\psi$  coefficients just as we would normally interpolate a constant value  $\phi_k$ .

In the special case where  $\hat{n}_k = 0$ , the normal constraints are exactly equivalent to the original value constraints. As a result we can easily mix constraints with and without normals.

In the case where we only use the constant basis function, so that  $\mathbf{b}(\mathbf{x}) = [1]$ , the fit from Equation (5) simplifies to

$$\begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) \\ \vdots \\ w(\mathbf{x}, \mathbf{p}_i) \end{bmatrix} c_1 = \begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) \\ \ddots \\ w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} S_1(\mathbf{x}) \\ \vdots \\ S_N(\mathbf{x}) \end{bmatrix} \quad (19)$$

which has the very intuitive interpretation that the interpolating function's value at  $\mathbf{x}$  is simply the weighted average of the values at  $\mathbf{x}$  predicted by each of the  $S_k(\mathbf{x})$ .

We have found this approach to work well. Figure 4 illustrates that the undesirable behavior that occurs with quasi-normal constraints does not occur with this method. Further, this approach causes the surface normals to actually take on the desired value at constraint points, whereas offset constraints do not. For polygonal constraints, the normals are interpolated so long as they are consistent with the polygon's plane. In addition to being useful with moving least-squares, this normal constraint approach should also work with other interpolation methods such as the radial splines used in [Turk and O'Brien, 1999]. Because the magnitude of the normal constraint grows linearly as the evaluation point moves away, we must choose a weighting function that falls off faster than linearly.

### 3.4 Interpolation and Approximation

When the weighting function parameter,  $\epsilon$ , is set to zero, the moving least-squares function will exactly interpolate constraint values. If we follow the general approach described

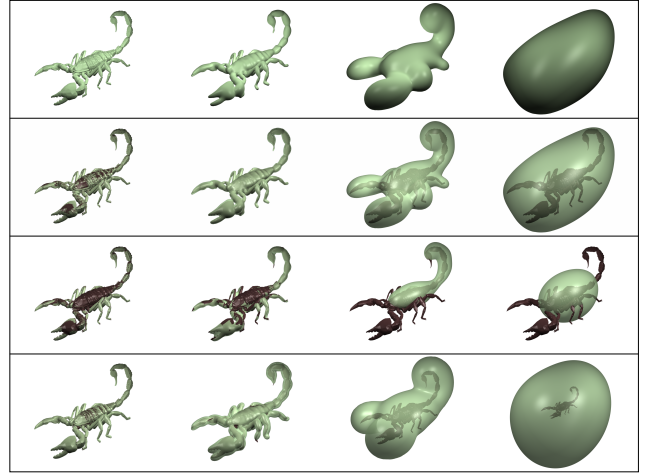


Figure 6: The first (top) row shows the result of applying our iterative adjustment algorithm with different values of  $\epsilon$  to a polygonal scorpion model. The second row shows the original and constructed surfaces together. The third row shows the result of only adjusting the surface to average values (no iterative adjustment). The fourth row shows the result generated when no correction is applied.

in [Turk and O'Brien, 1999] and [Ohtake et al., 2003a] of constraining the function to be zero at input points or polygons, supplying appropriate normal constraints, and extracting the iso-surface  $f(\mathbf{x}) = 0$ , then all the input polygons will be parts of the resulting implicit surface.

If the polygonal surface contains gaps or holes, then the implicit surface will extend beyond the input polygons to generate a closed surface. As with previous methods that accomplish hole filling using some form of implicit surface, there is no guarantee that the results will satisfy any particular criteria. However, we generally find that these extensions close gaps and holes in a useful fashion that produces results similar to what a human might have selected.

If the polygon surface self-intersects, then the interpolating surface will have some form of saddle at the intersections. This behavior is illustrated for a two-dimensional example in Figure 5.

When  $\epsilon$  is set to a non-zero value the weighting function is no longer singular at zero, and the moving least-squares function interpolates constraint values only approximately. Examination of Equation (4) reveals that  $\epsilon$  has the same units as distance. It corresponds to a feature size parameter: structures smaller than  $\epsilon$  tend to be smoothed away by the approximation.

While generating an approximate surface by simply setting  $\epsilon$  to some non-zero value works well to a limited extent, it suffers from two problems. The first is that as epsilon is set to larger values, the approximating surface has the tendency to move away from the input data (Figure 6, bottom row). For example, very large values of  $\epsilon$  will smooth an object to a simple sphere-like shape, but the sphere radius may be several times the original object's circumradius. The second problem is that we cannot ensure that all the object's original vertices fall inside the implicit surface, and for some applications this guarantee is important.

To correct the first problem we simply build a moving least-squares function with the desired  $\epsilon$ , sample its average value over the input polygons, and then extract a surface at that iso-value (Figure 6, third row). Although this procedure may at first appear to require substantial extra work, the additional work is actually not particularly significant. The majority of computation is spent extracting the iso-surface, and that task still only needs to be done once.

By adjusting the iso-value we achieve a surface that, on average, stays close to the input data, but with this construction we expect that roughly half the original vertices will fall outside the surface. To ensure that original vertices lie inside the surface, we iteratively adjust the  $\phi$  values assigned to the vertices.

Initially, the  $\phi$  values associated with each vertex are all zero and the  $\phi$  associated with each triangle is the constant zero as well. If a vertex,  $v$ , protrudes outside the iso-surface (i.e.,  $f(v) > 0$ ), we adjust its  $\phi$  value by  $-\gamma f(v)$  where  $\gamma$  is an adjustment rate parameter between zero and one (typically close to one). Once the vertices of a triangle have been assigned different values, we linearly interpolate  $\phi$  over the triangle when computing integrals. This adjustment process is done iteratively until no original vertex falls outside the iso-surface. The final surface is guaranteed to enclose all input vertices, as illustrated in the top two rows of Figure 6. As with adjusting the iso-value, the majority of computation is still spent extracting the iso-surface, and that task still only needs to be done once.

Variations on this iterative procedure for adjusting the  $\phi$  values could also be used to enforce other conditions. For example, it could be used to guarantee that all points are within some set distance of the iso-surface. Conditions could be tested at points other than the initial vertices, and the iterative procedure could also adjust the normal direction or magnitude associated with each constraint.

### 3.5 Fast Evaluation

Naïve implementation of the moving least-squares function would require work linear in the number of constraints for each function evaluation. For large data sets, this naïve approach is completely infeasible. A similar problem arises with the partition-of-unity method used in [Ohtake et al., 2003a]. They address the problem using a hierarchical evaluation scheme that caches approximations based on local neighborhoods. Because partition-of-unity and moving least-squares methods are essentially equivalent methods, their hierarchical evaluation scheme could be used with our method as well. We have, however, implemented a different evaluation scheme which we describe briefly.

We observe that the primary expense for evaluating the moving least-squares function is the cost of computing the sums and integrals for Equation (12). Were it not for the weighting function's dependence on  $x$ , the terms would be constant and the summation would only need to be computed once.

For terms that peak near the evaluation point, the weighting function changes rapidly. However, the weight function changes only slowly for far terms. We can approximate groups of the slowly changing far terms by first summing them and then multiplying by their average weight.

For our hierarchical scheme, we first store the input triangles in a K-D tree where each triangle is stored at one of the leaf nodes. We then compute the unweighted integrals, Equations (15) and (16), for each triangle and store them, along with the triangle's axis-aligned bounding box, in the leaf nodes. The interior nodes store the unweighted sums of their children's integrals/sums, and a bounding box that encloses the union of their children's bounds. We also store an area-weighted "center of mass" for each node.

To evaluate the contribution of a subtree, we test the evaluation point to see if it falls outside the subtree's bounding box by a distance greater than  $\lambda$  times the box's diameter. If it does, we use the sums stored at the subtree's root node with a weight computed using the distance between the node's center of mass and the evaluation point. If the evaluation point is not sufficiently distant, we recursively test the node's children. Only when we find that a leaf node fails our



Figure 7: The top left image shows a polygonized version the Utah teapot which contains holes (around lid and tip of spout), and intersecting parts (handle and spout with body). The top right image is a near-interpolating surface which fills the holes and removes intersecting surfaces. The bottom row contains photographs of physical models built on a fused deposition machine. The bottom right image shows a physical cutaway model.

distance test do we need to compute the weighted integral terms for that node.

This scheme was easy to implement using existing K-D tree collision detection code, and it allows us to work with models consisting of several hundred thousand triangles. The user can make a trade-off between speed and accuracy by adjusting  $\lambda$ . Our examples were generated with  $\lambda$  between 0.01 (when  $\epsilon = 0$ ) and 0.1 (when  $\epsilon$  is large).

### 3.6 Preprocessing

Although the methods we have described in previous sections cope reasonably well with intersecting geometry and layers of internal structure, it may still be useful to first remove some of these polygons. In particular, our algorithm will happily produce surfaces corresponding to internal structures, even if only an exterior shell was desired. In these cases, we can pre-process the input to remove polygons that are not visible from the exterior using methods such as those in [Nooruddin and Turk, 2003] and [Nooruddin and Turk, 2000].

The normal constraints depend on consistently oriented normals. Unfortunately, many polygon models may have normals that randomly point inward or outward. We force normals on topological surfaces to point in a consistent direction. We also orient the normals of any exterior-visible polygon to point outward. If both sides of a triangle are exterior-visible then we set that triangle's normal to zero.

## 4 Results and Discussion

Figures 1, 10 and 11 show the result of applying our algorithm to a variety of models using different values of  $\epsilon$ . Animations showing continuous variation of  $\epsilon$  from interpolating to extreme smoothing appear on the proceedings DVD. Most of these models contain holes, self-intersections, non-manifold structure, and other defects. The objects in brown are the original polygonal models. Green objects are output from our algorithm. For sufficiently large  $\epsilon$ , all objects converge to a circumscribing ellipsoid-like shape. As Figure 8 shows, the interpolating surfaces can reproduce small features and sharp edges.

As Figure 7 shows, we can use this algorithm as an effective preprocessor before sending a model to a rapid prototyping machine. The Utah teapot contains holes and self-intersections that would cause the machine to produce garbage output. A tightly approximating implicit surface does not contain those problems and allows a successful

Model	Fig.	P. In	$\epsilon$	V. Out	Time	$\epsilon$	V. Out	Time	$\epsilon$	V. Out	Time	$\epsilon$	V. Out	Time
Heavy Loader	1	37	0	2000	11:42	0.05	800	64:06	5	62	72:48	30	30	92:34
Teapot	10	6.3	0	1000	5:50	0.8	300	10:28	10	53	22:02	60	26	42:31
Cow	10	5.8	0	1000	5:37	0.8	300	8:04	7	61	23:35	120	23	58:19
Bunny	10	69	0	1500	8:13	0.4	400	19:34	10	50	41:36	60	28	72:23
Dragon	10	870	0	2000	12:23	0.6	400	82:21	7	46	89:54	30	21	97:04
Scorpion	10	78	0	1500	9:54	0.6	400	67:50	5	65	61:06	30	26	80:55
Intersecting Star	11	0.05	0	1000	4:44	1	300	5:18	10	48	7:03	110	28	8:20
Machine Part	11	0.8	0	1000	4:45	0.8	300	5:54	7	60	8:50	120	29	20:21
Deck Chair	11	3.9	0	1000	5:01	0.8	300	8:29	10	55	27:06	120	30	52:36
Armchair	11	3.4	0	1000	4:56	0.4	300	7:53	7	62	28:28	120	28	41:09
Cube Shape	11	0.01	0	1000	4:44	0.8	300	5:01	10	45	3:06	80	25	1:52

Table 1: This table lists the computation times and  $\epsilon$  parameter for the examples used in this paper. The columns P. In and V. Out list the number (in thousands) of polygons in the input model and the number (in thousands) of vertices in the output surface. We measure  $\epsilon$  in thousandths of the diagonal length of the object's bounding box. Computation times (minutes:seconds) measure total time on a 3 GHz P4 beginning with reading the input model and ending with writing the polygonized output surface. Column groups match the  $\epsilon$  values used for the examples shown in the figures.

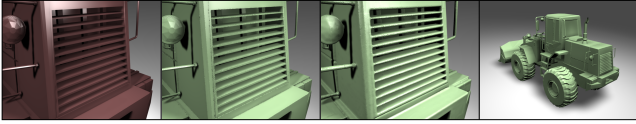


Figure 8: The far-left image shows a closeup view of the original polygons for the heavy-loader's back grill. The center-left image shows the resulting interpolating surface, and the center-right a slightly approximating one. The far-right image shows a rear view of the interpolating surface for the entire loader. The dented appearance near sharp edges is a polygonization artifact.

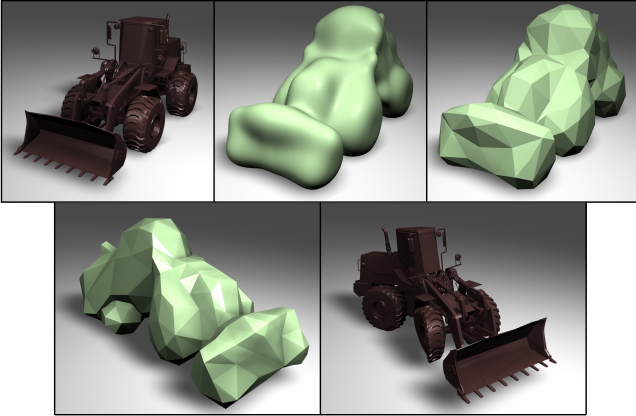


Figure 9: The heavy-loader shown top left contains many defects that make it unsuitable for simulation as a deformable object. The approximating surface, top center, fully encloses the original model. The tetrahedral finite-element model, top right, can be used as a simulation envelope to model the effect of an impact, lower left and lower right.

build. Additionally, because building a solid teapot would waste material, it is desirable to include an inner surface. We generated the inner surface of the cutaway teapot by taking the same MLS function used to create the outer surface and computing another iso-surface for a lower iso-value. These photographs demonstrate that our method produces surfaces that can be used to generate structurally sound physical models.

Deformable object simulations based on the finite element method have found widespread use in video games and film production. Unfortunately, self-intersections, topological inconsistencies, holes, and triangles with bad aspect ratios render most graphics models ill-suited for use as a finite element mesh. Even meshes that are free of these problems may contain far too many elements to be practical for simulation. We can still animate these objects by embedding them in

a suitable, enclosing, deformable mesh. As demonstrated by Figure 9, the tight, smooth, enclosing surfaces that can be generated with our method make excellent simulation envelopes.

Currently, we are using the polygonizer described in [Bloomenthal, 1994] for extracting iso-surfaces. It works well for smooth surfaces, but extracting small features requires a very fine resolution and produces models with an inordinate number of polygons. Our polygonal models produce useful envelopes after being passed through surface simplification software (see Figure 9), but extracting them is time consuming and requires substantial storage. (See Table 1.) We are currently considering better methods for surface extraction based on the algorithm from [Boissonnat and Oudot, 2003]. The surfaces for the heavy-loader shown in Figures 1 and 8 were extracted using a partial implementation of that algorithm.

## Acknowledgments

We thank the other members of the Berkeley Graphics Group for their helpful criticism and comments. We especially thank Ravi Kolluri for his help with polygonization, Adam Bargteil for help with the heavy-loader simulation, Carlo Séquin for his help with the fused deposition machine used to make the teapots, and Okan Arikan for help rendering. Images in this paper were rendered with Pixie. This work was supported in part by NSF CCR-0204377, California MICRO 02-055, and by generous support from Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, the Okawa Foundation, and the Alfred P. Sloan Foundation.

## References

- ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *IEEE Visualization 2001*, 21–28.
- ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2003. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (Jan.), 3–15.
- BELYTSCHKO, T., KRONGAUZ, Y., ORGAN, D., FLEMING, M., AND KRYSL, P. 1996. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering* 139, 3–47. Special issue on meshless methods.
- BITTAR, E., TSINGOS, N., AND GASCUEL, M.-P. 1995. Automatic reconstruction of unstructured 3d data: Combining a medial axis and implicit surfaces. *Proceedings of Eurographics 95*, 457–468.
- BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*. 324–349.



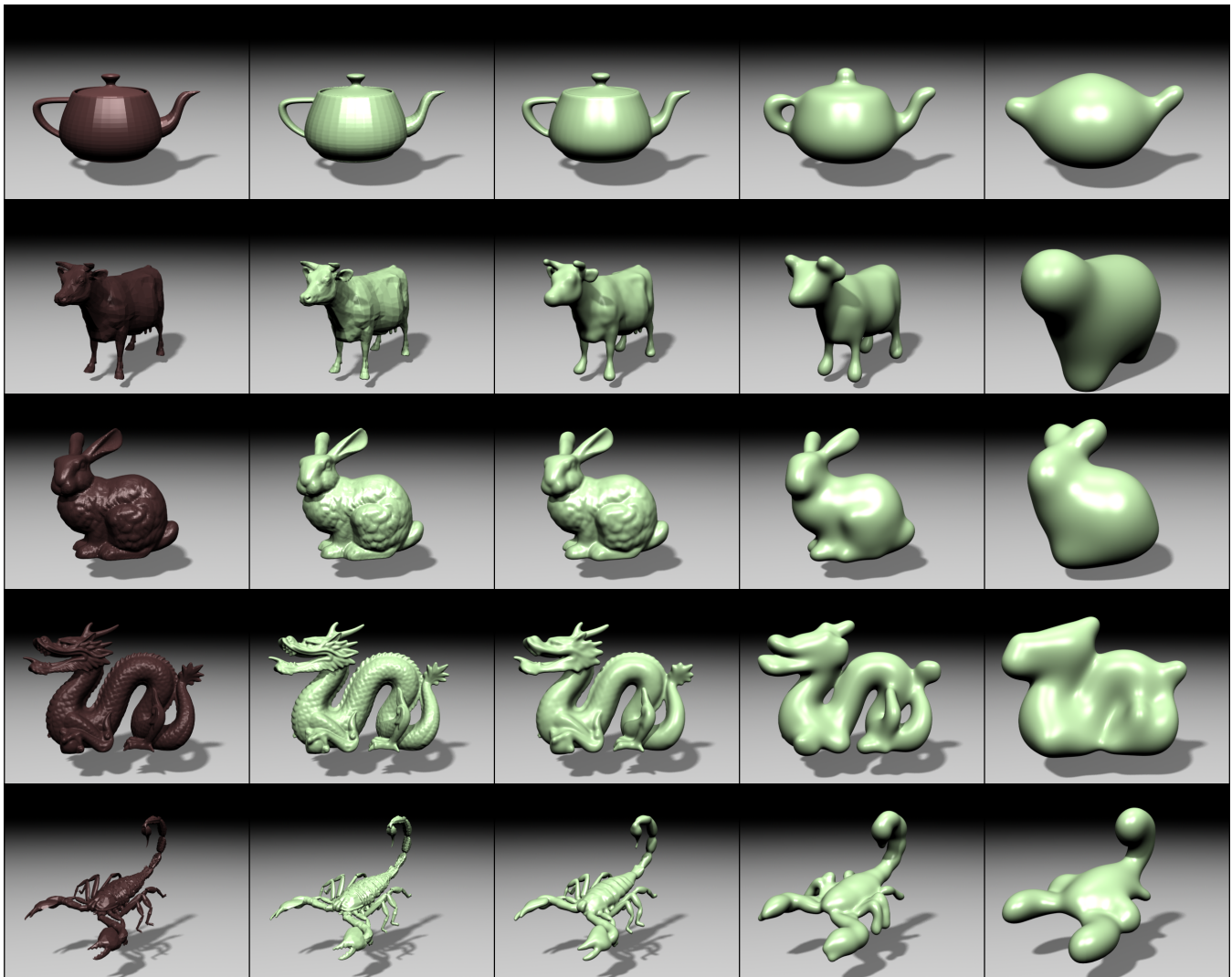


Figure 10: A collection of polygonal models processed with our algorithm. [Continued on next page.]

- BLOOMENTHAL, J., Ed. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., San Francisco, California.
- BOISSONNAT, J. D., AND OUDOT, S. 2003. Provably good surface sampling and approximation. In *Proceedings of the ACM SIGGRAPH Symposium on Geometry Processing*, 9–18.
- CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, 67–76.
- COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., JR., F. P. B., AND WRIGHT, W. 1996. Simplification envelopes. In *Proceedings of ACM SIGGRAPH 1996*, 119–128.
- COHEN-OR, D., SOLOMOVICI, A., AND LEVIN, D. 1998. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics* 17, 2 (Apr.), 116–141.
- DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 1999*, 317–324.
- FLEISHMAN, S., ALEXA, M., COHEN-OR, D., AND SILVA, C. T. 2003. Progressive point set surfaces. *ACM Transactions on Graphics* 22, 4 (Oct.), 97–1011.
- JONES, T. R., DURAND, F., AND DESBRUN, M. 2003. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics* 22, 3 (July), 943–949.
- KEREN, D., AND GOTSMAN, C. 1998. Tight fitting of convex polyhedral shapes. *International Journal of Shape Modeling*, 111–126.
- MORSE, B., YOO, T. S., RHEINGANS, P., CHEN, D. T., AND SUBRAMANIAN, K. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings of Shape Modelling International*, 89–98.
- MURAKI, S. 1991. Volumetric shape description of range data using “blobby model”. In *Proceedings of ACM SIGGRAPH 1991*, 227–235.
- MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. 2002. Level set surface editing operators. *ACM Transactions on Graphics* 21, 3 (July), 330–338.
- NOORUDDIN, F. S., AND TURK, G. 2000. Interior/exterior classification of polygonal models. In *IEEE Visualization 2000*, 415–422.
- NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (Apr.), 191–205.
- OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition of unity implicits. *ACM Transactions on Graphics* 22, 3 (July), 463–470.
- OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2003. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *Proceedings of Shape Modelling International*, 292–300.
- SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNII, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum* 14, 4 (Oct.), 181–188.

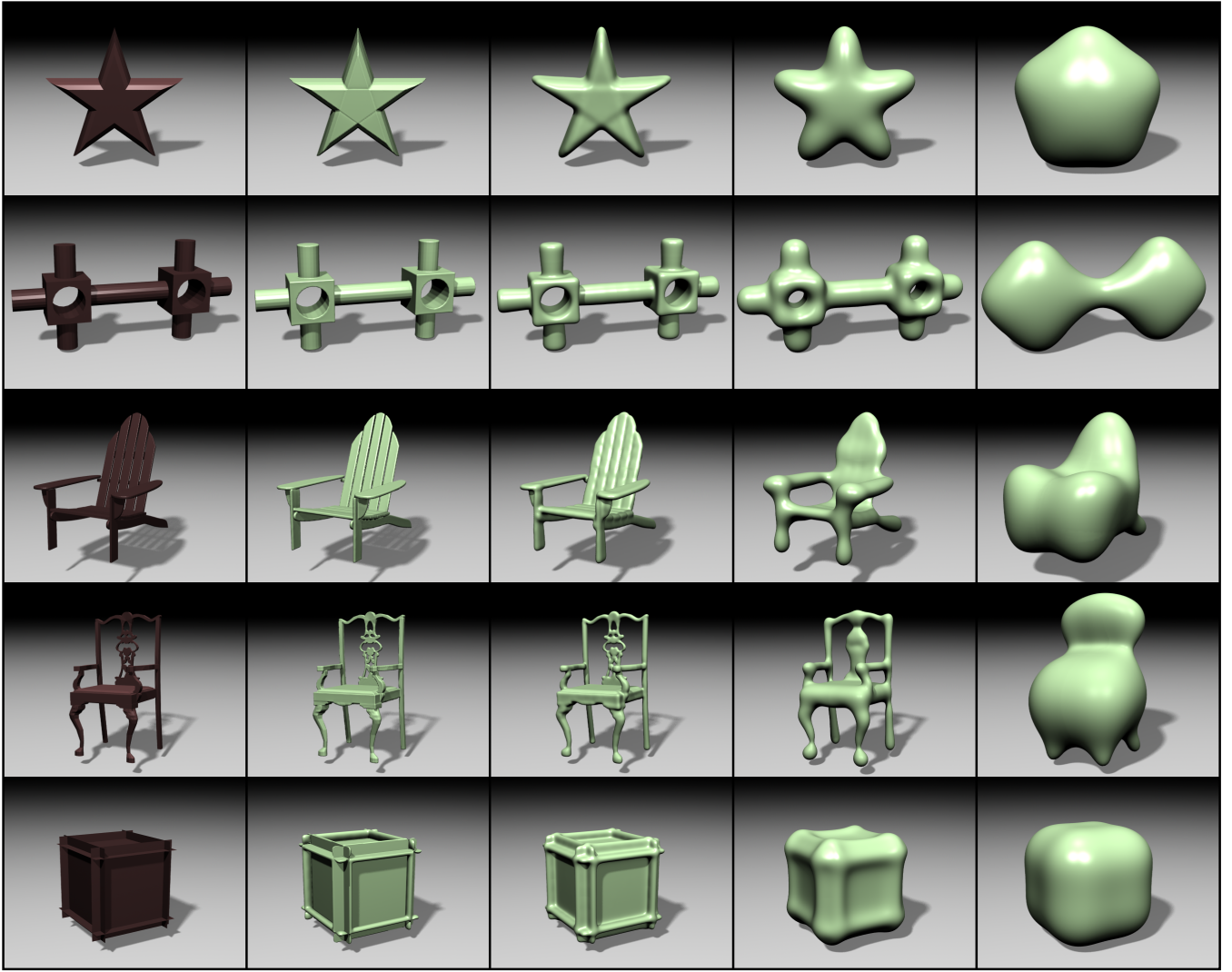


Figure 11: Additional polygonal models processed with our algorithm. [Continued from previous page.]

- TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 1995*, 351–358.
- TURK, G., AND O'BRIEN, J. F. 1999. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 1999*, 335–342.
- TURK, G., AND O'BRIEN, J. F. 2002. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics* 21, 4 (Oct.), 855–873.
- YNGVE, G., AND TURK, G. 2002. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics* 8, 4 (Oct.), 346–359.
- ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. In *IEEE Workshop on Variational and Level Set Methods*, 194–202.

## A Analytical Line Integrals

Our integrated constraints require solving integrals of the form

$$\int \frac{P(\mathbf{p})}{R(\mathbf{p})} d\mathbf{p} \quad (20)$$

where  $P(\mathbf{p})$  and  $R(\mathbf{p})$  are functions in  $\mathbf{p}$ . The functions  $P$  and  $R$  are respectively determined by the basis functions and the weighting function. For our choices,  $P$  is a constant or linear polynomial, and  $R$  is a quadratic polynomial with restricted form. We

cannot do the two-dimensional integral analytically, but the one-dimensional line integral one does have an analytic solution.

Once we have selected a direction for the line integration, the integrals for the constant and linear terms of  $P$  appear in the following forms:

$$\int_0^a \frac{1}{((x+k_1)^2+k_2)^2} dx \quad (21)$$

$$\int_0^a \frac{x}{((x+k_1)^2+k_2)^2} dx \quad (22)$$

where  $k_1$  and  $k_2$  are constant with respect to the integration variable,  $x$ .

The solutions to these integrals are

$$\beta \frac{-a\sqrt{k_2}(k_1(a+k_1)-k_2)-(k_1^2+k_2)((a+k_1)^2+k_2)}{2(k_2)^{\frac{3}{2}}(k_1^2+k_2)((a+k_1)^2+k_2)} \quad (23)$$

and

$$\beta \frac{a\sqrt{k_2}(a+k_1)+k_1((a+k_1)^2+k_2)}{2(k_2)^{\frac{3}{2}}((a+k_1)^2+k_2)} \quad (24)$$

respectively, where

$$\beta = \left( \tan^{-1} \left( \frac{k_1}{\sqrt{k_2}} \right) - \tan^{-1} \left( \frac{a+k_1}{\sqrt{k_2}} \right) \right) \quad (25)$$

# Provably Good Moving Least Squares

Ravikrishna Kolluri \*

## Abstract

We analyze a *moving least squares* algorithm for reconstructing a surface from point cloud data. Our algorithm defines an implicit function  $I$  whose zero set  $U$  is the reconstructed surface. We prove that  $I$  is a good approximation to the signed distance function of the sampled surface  $F$  and that  $U$  is geometrically close to and homeomorphic to  $F$ . Our proof requires sampling conditions similar to  $\epsilon$ -sampling, used in Delaunay reconstruction algorithms.

## 1 Introduction

Point sets have become a popular shape representation as current scanning devices generate dense point sets capable of modeling highly detailed surfaces. These point-based representations have several advantages in modeling and simulation, as mesh topology need not be maintained during surface deformations. However, a continuous definition of the surface represented by the points is needed for some applications such as rendering and resampling. Surface reconstruction algorithms are used to recover these smooth surfaces from point clouds.

The input to our surface reconstruction algorithm is a set  $S$  of sample points close to the surface  $F$  of a 3D object. Each sample point has an approximate surface normal. The output is an approximation of  $F$ . The approximation is represented either implicitly as the zero surface of a scalar function or as a surface triangulation.

Our surface reconstruction algorithm is based on a data interpolation technique called *moving least squares* (MLS). For each sample  $s \in S$  we define a globally smooth point function that approximates the signed distance from  $F$  in the local neighborhood of  $s$ . These functions are then blended together using Gaussian weight functions, yielding a smooth implicit function whose zero set is the reconstructed surface.

Our MLS construction is not new; it was originally proposed by Shen, O’Brien, and Shewchuk [25] for building manifold surfaces from polygon soup. The main contribution of this paper is to introduce theoretical guarantees for MLS algorithms. We prove that the implicit function generated by our algorithm is a good approximation of the signed distance function of the original surface. We also show that the reconstructed surface is geometrically and topologically correct.

The *crust* algorithm of Amenta and Bern [3] was the first surface reconstruction algorithm that guaranteed a correct reconstruction for sufficiently dense sample sets. The crust is defined as a subset of the faces in the Delaunay complex of the sample points. The sampling requirements are defined in

terms of *local feature size*, which is the distance from a point on the surface to its closest point on the medial axis. Our sampling conditions, defined in Section 3, are also based on the *local feature size*.

Unlike Delaunay-based algorithms, the MLS surface built by our algorithm might not interpolate the sample points. This allows us to reconstruct smooth surfaces from noisy point clouds. Our algorithm can handle noisy data as long as the amount of noise is small compared to the local feature size of the sample points.

## 2 Related Work

There has been much work on surface reconstruction from points clouds. A widely used technique defines the reconstructed surface as the zero set of a three-dimensional scalar function built from the input points. Hoppe, DeRose, Duchamp, McDonald, and Stuetzle [16] provide one of the earliest algorithms, which locally estimates the signed distance function induced by the “true” surface being sampled. Curless and Levoy [13] developed an algorithm that is particularly effective for laser range data comprising billions of point samples, like the statue of David reconstructed by the Digital Michelangelo Project [18].

Smooth surfaces can also be built by fitting globally supported basis functions to a point cloud. Turk and O’Brien [26] show that a global smooth approximation can be obtained by fitting radial basis functions. Carr et al. [12] adapt this radial basis function-fitting algorithm to large data sets using multipole expansions.

Instead of computing a single global approximation, moving least squares algorithms locally fit smooth functions to each sample point and blend them together. Ohtake, Belyaev, Alexa, Turk, and Seidel [22] use a partition-of-unity method with a fast hierarchical evaluation scheme to compute surfaces from large data sets. Our MLS construction is based on the algorithm given by Shen, O’Brien, and Shewchuk [25] that introduced the idea of associating functions, rather than just values, with each point to ensure that the gradient of the implicit function matches the gradient of the signed distance function near the sample points.

A different approach to moving least squares is the non-linear projection method originally proposed by Levin [17]. A point-set surface is defined as the set of stationary points of a projection operator. This surface definition was first used by Alexa et al. [2] for point based modeling and rendering. Since then the surface definition has been used for progressive point-set surfaces [15] and in PointShop3D [24], a point based modeling tool. Amenta and Kil [6] give an

\*Department of Computer Science, University of California, Berkeley, Berkeley, CA, 94720. Email: rkolluri@acm.org

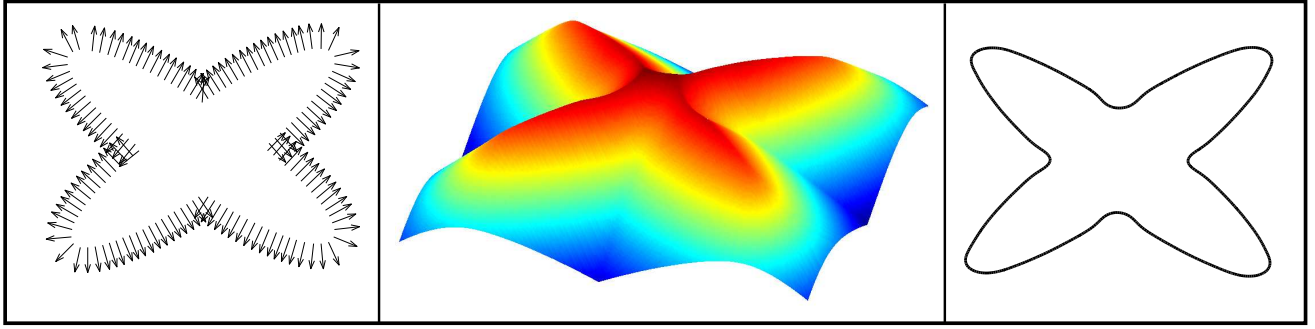


Figure 1: Left, a set of points with outside normals. Center, the implicit function built by our algorithm from the points. Right, the reconstructed curve which is the zero set of the implicit function.

explicit definition of point set surfaces as the local minima of an energy function along the directions given by a vector field. Adamson and Alexa [1] provide a simplified implicit surface definition for efficient ray tracing and define sampling conditions that guarantee a manifold reconstruction. However, current definitions of point-set surfaces come with no guarantees on the correctness of the reconstructed surface.

Following the crust algorithm of Amenta and Bern [3], there have been many Delaunay-based algorithms for surface reconstruction with provable guarantees. Amenta, Choi, Dey, and Leekha [4] present the cocone algorithm, which is much simpler than the crust and prove that the reconstructed surface is homeomorphic to the original surface. The powercrust algorithm of Amenta, Choi, and Kolluri [5] uses weighted Delaunay triangulations to avoid the manifold extraction step of the crust and cocone algorithms. Boissonnat and Cazals [8] build a smooth surface by blending together functions associated with each sample point, using natural neighbor coordinates derived from the Voronoi diagram of the sample points. The robust cocone algorithm of Dey and Goswami [14] guarantees a correct reconstruction for noisy point data. Even when the input points are noisy, surfaces reconstructed by Delaunay algorithms interpolate (a subset of) the sample points. As a result, the reconstructed surface is not smooth and a mesh smoothing step is often necessary.

Smooth meshes that approximate  $F$  can be built by contouring the zero set of the implicit function defined by our algorithm. The marching cubes [19] algorithm is widely used for contouring level sets of implicit functions. There has been some recent work on contouring algorithms with theoretical guarantees. Boissonnat and Oudot [11] give a Delaunay-based contouring algorithm that guarantees good quality triangles in the reconstructed surface. Boissonnat, Cohen-Steiner and Vegter [10] present a contouring algorithm with guarantees on the topology of the reconstructed triangulation.

Signed distance functions of surfaces are useful in their own right. Level set methods that have been used in surface reconstruction [27], physical modeling of fluids, and in many other areas rely on signed distance functions to implicitly

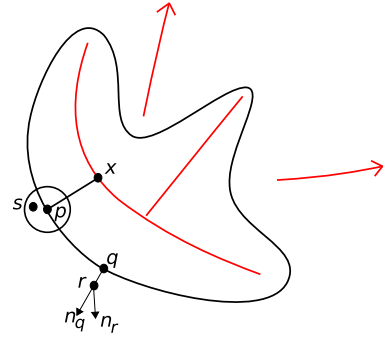


Figure 2: A closed curve along with its medial axis. The local feature size of  $p$  is the distance to the closest point  $x$  on the medial axis.

maintain moving surfaces. See the book by Osher and Fedkiw [23] for an introduction to level set methods. Mitra et al. [20] use approximation of signed distance functions to align overlapping surfaces. The implicit function constructed by our algorithm can be used as an approximation to the signed distance function.

### 3 Sampling Requirements

The *local feature size* (lfs) at a point  $p \in F$  is defined as the distance from  $p$  to the nearest point of the medial axis of  $F$ . A point set  $S$  is an  $\epsilon$ -sample if the distance from any point  $p \in F$  to its closest sample in  $S$  is less than  $\epsilon \text{lfs}(p)$ . Amenta and Bern [3] prove that a good approximation to  $F$  can be obtained from the Delaunay triangulation of an  $\epsilon$ -sample  $S$ .

Our results on the correctness of the reconstructed surface require *uniform  $\epsilon$ -sampling*. Assume that the data set has been scaled such that the lfs of any point on  $F$  is at least 1. A point set  $S$  is a *uniform  $\epsilon$ -sample* of  $F$  if the distance from each point  $p \in F$  to its closest sample is less than  $\epsilon$ . The amount of noise in the samples should be small compared to the sampling density. For each sample  $s$ , the distance to its closest surface point  $p$  should be less than  $\epsilon^2$  as shown in Figure 2. Moreover, the angle between the normal  $\vec{n}_r$  of a sample  $r$  and the normal  $\vec{n}_q$  of the closest surface point to  $r$ , should be less than  $\epsilon$ .



Arbitrary oversampling in one region of the surface can distort the value of the implicit function in other parts of the surface. As this rarely happens in practice, we assume that local changes in the sampling density are bounded. Let  $\alpha_p$  be the number of samples inside a ball of radius  $\epsilon$  centered at a point  $p$ . We assume that for each point  $p$ , if  $\alpha_p > 0$ , the number of samples inside a ball at radius  $2\epsilon$  at  $p$  is at most  $8\alpha$ . We will use two parameters to state our geometric results. The value of  $\epsilon$  depends on the sampling density and we define a second parameter,  $\tau = 2\epsilon$ . The results in this paper hold true for values of  $\epsilon \leq 0.01$ .

#### 4 Surface Definition

Given a set of sample points  $S$  near a smooth, closed surface  $F$ , our algorithm builds an implicit function  $I$  whose zero surface  $U$  approximates  $F$ . We assume that the outside surface normal  $\vec{n}_i$  is approximately known for each sample point  $s_i \in S$  as shown in Figure 1. In practice, the normal of  $s_i$  is obtained by local least squares fitting of a plane to the samples in the neighborhood around  $s_i$ . Mitra, Nguyen and Guibas [21] analyze the least squares method for normal estimation and present an algorithm for choosing an optimal neighborhood around each sample point.

Our algorithm begins by constructing a point function for each sample point in  $S$ . The point function  $P_{s_i}$  for sample point  $s_i$  is defined as the signed distance function from  $x$  to the tangent plane at  $s_i$ ,  $P_{s_i}(x) = (x - s_i) \cdot \vec{n}_i$ .

The implicit function  $I$  is a weighted average of the point functions.

$$I(x) = \frac{\sum_{s_i \in S} W_i(x)((x - s_i) \cdot \vec{n}_i)}{\sum_{s_j \in S} W_j(x)}.$$

We use Gaussian functions,  $W_i(x) = e^{-\|x - s_i\|^2 / \epsilon^2} / A_i$  in computing the weighted average of the point functions. Here  $A_i$  is the number of samples inside a ball of radius  $\epsilon$  centered at  $s_i$ , including  $s_i$  itself.

#### 5 Geometric Properties

Amenta and Bern [3] prove the following Lipschitz condition on the surface normal with respect to the *local feature size*. As we assume that for each point  $p \in F$ ,  $\text{lfs}(p) \geq 1$ , we can state the Lipschitz condition in terms of the distance between two points.

**Theorem 1.** *For points  $p, q$  on the surface  $F$  with  $d(p, q) \leq r$ , for any  $r < 1/3$ , the angle between the normals at  $p$  and  $q$  is at most  $r/(1 - 3r)$  radians.*

Consider the surface inside a small ball  $B$  centered at a point  $p \in F$ . As shown in Figure 3, the surface inside  $B$  has to be outside the medial balls at  $p$ . As a result, the surface inside  $B$  lies between two planes close to the tangent plane of  $p$ .

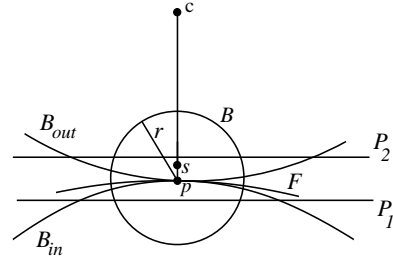


Figure 3: The surface inside a ball  $B$  of radius  $r$  has to be outside the medial balls  $B_i$  and  $B_o$ . As a result, all samples in  $B$  are between two planes  $P_1$  and  $P_2$  that are at a distance of  $O(r^2 + \epsilon^2)$  from  $p$ .

**Observation 2.** *For a point  $p \in F$ , let  $B$  be a ball of radius  $r < 0.1$  centered at  $p$ . The samples inside  $B$  lie between two planes  $P_1, P_2$  parallel to the tangent plane at  $p$ . The distance from  $p$  to  $P_1, P_2$  is less than  $\frac{(r + \epsilon^2)^2}{2} + \epsilon^2$ .*

Let  $F_{\text{out}}$  be the outside  $\tau$ -offset surface of  $F$  that is obtained by moving each point  $x$  on  $F$  along the normal at  $x$  by a distance  $\tau$ . Similarly, let  $F_{\text{in}}$  be the inside  $\tau$ -offset surface of  $F$ . The  $\tau$ -neighborhood is the region bounded by the inside and the outside offset surfaces. For any point  $x$  inside the  $\tau$ -neighborhood,  $|\phi(x)| < \tau$ , where  $\phi$  is the signed distance function of  $F$ .

The main result in this section is that the zero set  $U$  is geometrically close to  $F$ . We show this by proving that  $U$  is inside the  $\tau$ -neighborhood of  $F$  (Theorem 9). We then show that the reconstructed surface is a manifold by proving that the gradient of  $I$  is non-zero at each point in the zero set of  $I$  (Theorem 18).

Consider a point  $x$  shown in Figure 4, whose closest point on the surface is  $p$ . The vector  $\vec{x}p$  is parallel to the surface normal of  $p$  and  $\|\vec{x}p\| = |\phi(x)|$ . Let  $B_1(x), B_2(x)$  be two balls centered at point  $x$ . The radius of  $B_1(x)$  is  $|\phi(x)|$  and  $B_2(x)$  is a slightly larger ball whose radius is  $|\phi(x)| + \tau + \epsilon$ .

Let  $N(x)$  be the weighted combination of the point functions at  $x$ ,  $N(x) = \sum_{s_i \in S} W_i(x)P_{s_i}(x)$  and let  $D(x)$  be the sum of all weight functions at  $x$ ,  $D(x) = \sum_{s_i \in S} W_i(x)$ .

Our geometric results are based on the observation that samples outside  $B_2(x)$  have little effect on the value of the implicit function at  $x$ . To see this, we first separate the contributions of samples inside  $B_2(x)$  and samples outside  $B_2(x)$ . Let  $N_{\text{in}}(x) = \sum_{s_i \in B_2(x)} W_i(x)P_{s_i}(x)$  and  $N_{\text{out}}(x) = \sum_{s_i \notin B_2(x)} W_i(x)P_{s_i}(x)$  be the contributions to  $N(x)$  by samples inside and outside  $B_2(x)$ . Similarly, let  $D_{\text{in}}(x) = \sum_{s_i \in B_2(x)} W_i(x)$  and  $D_{\text{out}}(x) = \sum_{s_i \notin B_2(x)} W_i(x)$  be the contributions to  $D(x)$  by samples inside and outside  $B_2(x)$ .

Consider the space outside  $B_2(x)$  divided into spherical shells of width  $\epsilon$  as shown in Figure 4. Let  $H_w$  be the region between balls of radius  $w$  and  $w + \epsilon$ . We bound the contributions of all samples outside  $B_2(x)$  by summing over the contributions of all samples in each shell.

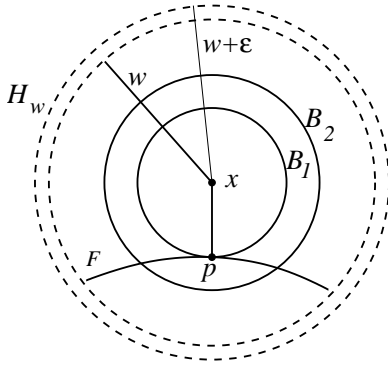


Figure 4: For a point  $x$ ,  $p$  is the closest point to  $x$  on the surface. The space outside the ball  $B_2(x)$  is divided into spherical shells of width  $\epsilon$ .  $H_w$  is the shell bounded by spheres of radius  $w$  and  $w + \epsilon$ .

We begin by proving an upper bound on the number of samples inside each shell normalized by their oversampling factors.

**Lemma 3.** For a ball  $B_\epsilon$  of radius  $\frac{\epsilon}{2}$ ,  $\sum_{s_i \in B_\epsilon} \frac{1}{A_i} \leq 1$ .

*Proof.* If  $B_\epsilon$  is empty we are done; assume that  $B_\epsilon$  contains  $\alpha > 0$  samples. Let  $s_i$  be a sample inside  $B_\epsilon$ . As all samples inside  $B_\epsilon$  are inside a ball of radius  $\epsilon$  centered at  $s_i$ ,  $A_i \geq \alpha$ . Hence the contribution of all samples inside  $B_\epsilon$  is given by  $\sum_{s_i \in B_\epsilon} \frac{1}{A_i} \leq \alpha \frac{1}{\alpha} \leq 1$ .  $\square$

**Lemma 4.** For samples  $s_i$  in spherical shell  $H_w$  centered at point  $x$ ,

$$\sum_{s_i \in H_w} \frac{1}{A_i} < \frac{200}{\epsilon^2} (w^2 + w\epsilon + \epsilon^2).$$

*Proof.* Let  $C$  be the smallest number of spheres of radius  $\frac{\epsilon}{2}$  that cover  $H_w$ . Consider a covering of  $H_w$  with axis-parallel cubes of size  $\frac{\epsilon}{\sqrt{3}}$ . Any cube that intersects  $H_w$  is inside a slightly larger shell bounded by spheres of radius  $w + 2\epsilon$  and  $w - \epsilon$  centered at  $x$ . So the number of cubes that cover  $H_w$  is less than  $\frac{36\sqrt{3}\pi\epsilon(w^2 + w\epsilon + \epsilon^2)}{\epsilon^3}$ . Any cube in this grid is covered by a sphere of radius  $\frac{\epsilon}{2}$ . Applying Lemma 3 to each sphere,  $\sum_{s_i \in H_w} \frac{1}{A_i} \leq C \leq \frac{36\sqrt{3}\pi\epsilon(w^2 + w\epsilon + \epsilon^2)}{\epsilon^3} < \frac{200}{\epsilon^2} (w^2 + w\epsilon + \epsilon^2)$ .  $\square$

**5.1 Offset Regions.** In this section we obtain bounds on the difference between  $I(x)$  and the signed distance function  $\phi(x)$  for points  $x$  outside the  $\tau$ -neighborhood and use these bounds to show that the implicit function is non-zero outside the  $\tau$ -neighborhood.

We begin with a result about the point functions of samples inside  $B_2(x)$ . We prove that for any sample  $s \in B_2(x)$ ,  $P_s(x)$  is close to  $\phi(x)$ . In order to state this result for points in the inside and outside offset regions, it is convenient to define  $\mu(x) = \frac{\phi(x)}{|\phi(x)|}$  to be the sign function of  $F$ . For  $x$  outside  $F_{\text{out}}$ ,  $\mu(x) = 1$  and for  $x$  inside  $F_{\text{in}}$ ,  $\mu(x) = -1$ .

**Lemma 5.** Let  $x$  be a point outside the  $\tau$ -neighborhood. Let  $P_s(x)$  be the point function of sample  $s \in B_2(x)$ , evaluated at  $x$ . Then,

$$\mu(x)P_s(x) \leq \mu(x)\phi(x) + 3\epsilon,$$

and,

$$\mu(x)P_s(x) \geq \mu(x)\phi(x)(1 - 6\epsilon) - 12\epsilon^2.$$

*Proof.* Let  $p$  be the closest point to  $x$  on the surface. Then,  $d(x, p) = \mu(x)\phi(x)$ . Recall that  $\tau = 2\epsilon$ . As  $s \in B_2(x)$ ,

$$\begin{aligned} \mu(x)P_s(x) &\leq d(x, s) \\ &\leq d(x, p) + \tau + \epsilon \\ &< \mu(x)\phi(x) + 3\epsilon. \end{aligned}$$

Let  $p'$  be the closest point to  $s$  on the surface  $F$  as shown in Figure 5. Let  $B_m$  be the medial ball touching  $p'$  on the side of  $F$  opposite  $x$  and let  $l$  be the radius of  $B_m$ . Let  $\theta$  be the angle between  $xp'$  and the normal at  $p'$ . The distance between  $x$  and the center of  $B_m$  is given by,

$$d^2(x, q) = (d(x, p') \cos \theta + l)^2 + d^2(x, p') \sin^2 \theta.$$

The medial ball  $B_m$  cannot intersect  $B_1(x)$  which is contained in a medial ball on the opposite side of the surface. Hence the sum of their radii should be less than the distance between their centers.

$$\begin{aligned} (l + d(x, p'))^2 &\leq (d(x, p') \cos \theta + l)^2 \\ &\quad + d^2(x, p') \sin^2 \theta. \\ \cos \theta &\geq \frac{2ld(x, p) - (d^2(x, p') - d^2(x, p))}{2ld(x, p')}. \end{aligned}$$

From the sampling conditions,  $d(x, s) \geq d(x, p') - \epsilon^2$  and the angle between the normal at  $s$  and the surface normal at  $p'$  is less than  $\epsilon$ . As  $d(x, p') \geq \tau$ , the angle between  $xs$  and  $xp'$  is less than  $\arcsin(\frac{\epsilon^2}{\tau}) < \frac{2\epsilon^2}{\tau}$ . So the angle between  $xs$  and the normal at  $s$  is at most  $\theta + \frac{2\epsilon^2}{\tau} + \epsilon$ . Using standard trigonometric formulae, it is easy to show that,  $\cos(\theta + \frac{2\epsilon^2}{\tau} + \epsilon) \geq \cos \theta - \frac{2\epsilon^2}{\tau} - \epsilon$ .

$$\begin{aligned} \mu(x)P_s(x) &\geq d(x, s) \cos(\theta + \frac{2\epsilon^2}{\tau} + \epsilon) \\ &\geq (d(x, p') - \epsilon^2) \cdot \\ &\quad \left( \frac{2ld(x, p) - (d^2(x, p') - d^2(x, p))}{2ld(x, p')} \right. \\ &\quad \left. - \frac{2\epsilon^2}{\tau} - \epsilon \right). \end{aligned}$$

From the local feature size assumption,  $l \geq 1$ . Substituting the value of  $\tau = 2\epsilon$  we have,

$$\mu(x)P_s(x) \geq \mu(x)\phi(x)(1 - 6\epsilon) - 12\epsilon^2. \quad \square$$

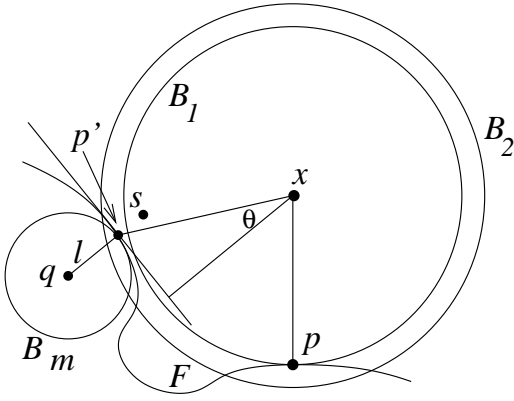


Figure 5: Sample  $s$  is inside  $B_2(x)$  and  $p'$  is the point closest to  $s$  on  $F$ .  $B_m$  is a medial ball touching  $p'$  on the side of  $F$  opposite  $x$ .

Using the result of Lemma 5, it is easy to show that for each sample  $s \in B_2(x)$ ,  $P_s(x)$  and  $\phi(x)$  are either both positive or both negative.

**Corollary 6.** *Let  $x$  be a point outside the  $\tau$ -neighborhood. For a sample  $s \in B_2(x)$ , let  $P_s(x)$  be the point function of  $s$  evaluated at  $x$ . For values of  $\epsilon \leq 0.08$ ,  $\phi(x)P_s(x) > 0$ .*

*Proof.* For a point  $x$  outside  $F_{\text{out}}$ ,  $\phi(x) \geq \tau \geq 2\epsilon$ . Applying the lower bound on  $P_s(x)$  from Lemma 5, we can write

$$\phi(x)P_s(x) \geq 2\epsilon(2\epsilon(1 - 6\epsilon) - 12\epsilon^2).$$

It is now easy to check that for  $\epsilon \leq 0.08$ ,  $2\epsilon(1 - 6\epsilon) - 12\epsilon^2 > 0$ . A similar argument proves the result for points inside  $F_{\text{in}}$ .  $\square$

We now prove two results showing that the points outside  $B_2(x)$  have little effect on the value of  $I(x)$ . In Lemma 7 we prove that  $D_{\text{out}}(x) \ll D_{\text{in}}(x)$  and in Lemma 8 we prove that  $|N_{\text{out}}(x)| \ll |N_{\text{in}}(x)|$ . We will use two constants  $c_1 = 0.05$  and  $c_2 = 0.01c_1$  to state our geometric results.

**Lemma 7.** *Let  $x$  be a point outside the  $\tau$ -neighborhood. Let  $D_{\text{out}}(x)$  and  $D_{\text{in}}(x)$  be the total weights of samples inside  $B_2(x)$  and outside  $B_2(x)$  at  $x$ , respectively. Then,  $\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} < c_1$ .*

*Proof.* Consider the division of space outside  $B_2(x)$  into spherical shells of width  $\epsilon$  starting with  $B_2(x)$  whose radius is  $w_0 = |\phi(x)| + \tau + \epsilon$ , as shown in Figure 4. The value of the Gaussian function associated with each sample inside shell  $H_w$  at  $x$  is at most  $e^{-w^2/\epsilon^2}$ . Using the bound on the weight of samples in  $H_w$  from Lemma 4,

$$\begin{aligned} D_{\text{out}}(x) &\leq \frac{200}{\epsilon^2} \sum_{i=0}^{\infty} (w_i^2 + \epsilon w_i + \epsilon^2) e^{-w_i^2/\epsilon^2} \\ &\leq \frac{200}{\epsilon^2} \sum_{i=0}^{\infty} (w_i^2 + \epsilon w_i + \epsilon^2) e^{-(w_0 w_i)/\epsilon^2}. \end{aligned} \quad (1)$$

Here  $w_i = w_0 + i\epsilon$  is the radius of the smaller sphere bounding the  $i^{\text{th}}$  shell. The dominant term in Equation 1 is a geometric series with a common ratio  $e^{-w_0/\epsilon} < 0.01$ . The summation has a closed form solution easily obtained using Mathematica. An upper bound is given by

$$D_{\text{out}}(x) \leq \frac{400}{\epsilon^2} (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-w_0^2/\epsilon^2}.$$

We now obtain a lower bound on the contribution of samples inside  $B_2(x)$ . Let  $B_\epsilon$  be a ball of radius  $\epsilon$  centered at  $p$ . From the sampling conditions, we know that  $B_\epsilon$  contains  $\alpha \geq 1$  samples. In our sampling requirements, we also assumed that the rate of change in sampling density is bounded. Since a ball of radius  $\epsilon$  centered at  $p$  contains  $\alpha \geq 1$  samples, a ball of radius  $2\epsilon$  centered at  $p$  contains at most  $8\alpha$  samples. So the normalizing factor associated with  $s_i \in B_\epsilon$ ,  $\frac{1}{A_i} \geq \frac{1}{8\alpha}$ . We now have a lower bound on the weight of samples inside  $B_\epsilon$ :

$$\begin{aligned} D_{\text{in}}(x) &\geq \sum_{s_i \in B_\epsilon} \frac{1}{A_i} e^{-(|\phi(x)| + \epsilon)^2/\epsilon^2} \\ &\geq \frac{\alpha}{8\alpha} e^{-(|\phi(x)| + \epsilon)^2/\epsilon^2} \\ &= \frac{1}{8} e^{-(|\phi(x)| + \epsilon)^2/\epsilon^2}. \end{aligned} \quad (2)$$

An upper bound on the ratio of the inside and the outside weights is given by,

$$\begin{aligned} \frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} &\leq \frac{3200}{\epsilon^2} (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-(w_0^2 - (|\phi(x)| + \epsilon)^2)/\epsilon^2} \\ &= \frac{3200}{\epsilon^2} (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-\tau(2|\phi(x)| + \tau + 2\epsilon)/\epsilon^2}. \end{aligned} \quad (3)$$

For  $|\phi(x)| \geq \tau$ ,  $\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)}$  is a monotonically decreasing function of  $|\phi(x)|$ . The maximum value is obtained for  $|\phi(x)| = \tau$  and  $w_0 = 2\tau + \epsilon = 5\epsilon$ .

$$\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} \leq \frac{3200}{\epsilon^2} (31\epsilon^2) e^{-16} < c_1. \quad \square$$

Using the proof technique of Lemma 7 we now prove that  $\frac{|N_{\text{out}}(x)|}{|N_{\text{in}}(x)|}$  is very small.

**Lemma 8.** *Let  $x$  be a point outside the  $\tau$ -neighborhood. Let  $N_{\text{out}}(x)$  and  $N_{\text{in}}(x)$  be the contributions of samples inside  $B_2(x)$  and outside  $B_2(x)$  to  $N(x)$ , respectively. Then,  $\frac{|N_{\text{out}}(x)|}{|N_{\text{in}}(x)|} < c_1$ .*

*Proof.* To compute an upper bound for  $N_{\text{out}}(x)$ , again consider the space outside  $B_2(x)$  divided into shells of radius  $\epsilon$  as shown in Figure 4. For  $w \geq \tau$ , the value of  $w e^{-w^2/\epsilon^2}$  decreases as  $w$  increases. Hence for a sample  $s_i$  inside the shell  $H_w$ ,  $|P_{s_i}(x)| W_i(x) \leq \frac{w e^{-w^2/\epsilon^2}}{A_i}$ . Let  $w_0 = |\phi(x)| + \tau + \epsilon$  and  $w_i = w_0 + i\epsilon$ . Using the bound on

the weight of samples in  $H_w$  from Lemma 4,

$$\begin{aligned} |N_{\text{out}}(x)| &\leq \frac{200}{\epsilon^2} \sum_{i=0}^{\infty} (w_i^2 + \epsilon w_i + \epsilon^2) w_i e^{-w_i^2/\epsilon^2} \\ &\leq \frac{200}{\epsilon^2} \sum_{i=0}^{\infty} (w_i^2 + \epsilon w_i + \epsilon^2) w_i e^{-(w_0 w_i)/\epsilon^2}. \end{aligned}$$

Like the summation in Equation 1, the above summation has a closed form solution easily obtained using Mathematica.

$$|N_{\text{out}}(x)| < \frac{400}{\epsilon^2} w_0 (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-w_0^2/\epsilon^2}.$$

From Corollary 6, the point functions of all samples inside  $B_2(x)$  have the same sign at  $x$ . Hence a lower bound on the contribution of the sample points inside  $B_2(x)$  to  $|N_{\text{in}}(x)|$  is given by summing over the contributions of all samples inside a ball  $B_\epsilon$  of radius  $\epsilon$  around  $p$ .

$$|N_{\text{in}}(x)| \geq \min_{s_i \in B_\epsilon(x)} \{|P_{s_i}(x)|\} D_{\text{in}}(x).$$

From Theorem 1, the angle between the normal of each sample  $s_i \in B_\epsilon(x)$  and the normal of  $p$  is at most  $2\epsilon$ . As  $x$  is outside the  $\tau$ -neighborhood,  $|P_{s_i}(x)| \geq \tau \cos 2\epsilon$ . Substituting the lower bound of  $D_{\text{in}}(x)$  from Equation 2,

$$|N_{\text{in}}(x)| > \frac{\tau}{10} e^{-(|\phi(x)|+\epsilon)^2/\epsilon^2}.$$

$$\begin{aligned} \frac{|N_{\text{out}}(x)|}{|N_{\text{in}}(x)|} &\leq \frac{4000}{\epsilon^2 \tau} w_0 (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-(w_0^2 - (|\phi(x)|+\epsilon)^2)/\epsilon^2} \\ &\leq \frac{4000}{\epsilon^2 \tau} w_0 (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-\tau(2|\phi(x)|+\tau+2\epsilon)/\epsilon^2}. \end{aligned}$$

The value of  $\frac{N_{\text{out}}(x)}{N_{\text{in}}(x)}$  is smallest for  $|\phi(x)|=\tau$ . Substituting the value of  $\tau = 2\epsilon$ ,

$$\frac{|N_{\text{out}}(x)|}{|N_{\text{in}}(x)|} \leq \frac{4000}{2\epsilon^3} (135\epsilon^3) e^{-16} < c_1. \quad \square$$

Lemma 8 proves that  $I(x)$  is mostly determined by the point functions of the samples inside the ball  $B_2(x)$ . We can derive bounds for  $I(x)$  in terms of  $\phi(x)$  by combining the results in Lemma 8 and Lemma 5.

$$\mu(x)I(x) \leq (\mu(x)\phi(x) + 3\epsilon)(1 + c_1). \quad (4)$$

We can also derive a similar lower bound on  $I(x)$ .

$$\mu(x)I(x) \geq (\mu(x)\phi(x)(1 - 6\epsilon) - 12\epsilon^2) \frac{1 - c_1}{1 + c_1}. \quad (5)$$

Equation 4 and Equation 5 show that the implicit function is close to the signed distance function for points outside the  $\tau$ -neighborhood. We now have all the tools required to prove our main geometric result: the implicit function  $I(x)$  is non-zero outside the  $\tau$ -neighborhood.

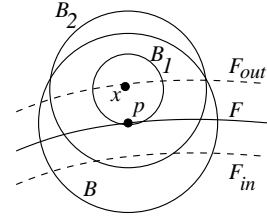


Figure 6: For  $x$  in the  $\tau$ -neighborhood, the samples inside balls  $B_1(x)$  and  $B_2(x)$  are inside a ball of radius  $6\epsilon$  centered at  $p$ .

**Theorem 9.** Let  $\epsilon \leq 0.08$ . For each point  $x$  outside  $F_{\text{out}}$ ,  $I(x) > 0$  and for each point  $y$  inside  $F_{\text{in}}$ ,  $I(y) < 0$ .

*Proof.* This proof is exactly like the proof of Corollary 6. It is easier to get the result directly from Equation 5. Consider a point  $x$  outside  $F_{\text{out}}$ . From Equation 5,

$$\begin{aligned} I(x) &\geq (\mu(x)\phi(x)(1 - 6\epsilon) - 12\epsilon^2) \frac{1 - c_1}{1 + c_1} \\ &\geq (2\epsilon(1 - 6\epsilon) - 12\epsilon^2) \frac{1 - c_1}{1 + c_1}. \end{aligned}$$

For  $\epsilon \leq 0.08$ , it is easy to check that  $2\epsilon(1 - 6\epsilon) - 12\epsilon^2 > 0$ . A similar argument proves that the implicit function is negative at any point inside  $F_{\text{in}}$ .  $\square$

Theorem 9 proves that the implicit function  $I$  does not have any spurious zero crossings far away from the sample points. We now have an upper bound of  $\tau$  on the Hausdorff distance between  $F$  and  $U$ .

**Theorem 10.** For a point  $x \in U$ , let  $p$  be the closest point in  $F$ . Then  $d(x, p) \leq \tau$ .

*Proof.* From Theorem 9, the implicit function has a non-zero value outside the  $\tau$ -neighborhood. Hence, the point  $x$  is constrained to lie inside the  $\tau$ -neighborhood of  $F$  and  $d(x, p) \leq \tau$ .  $\square$

**Theorem 11.** For a point  $p \in F$ , let  $x$  be the closest point in  $U$ . Then,  $d(x, p) \leq \tau$ .

*Proof.* If  $I(p) = 0$  we are done; assume without loss of generality that  $I(p) < 0$ . Let  $q$  be the closest point to  $p$  on the outside offset surface  $F_{\text{out}}$ . From Theorem 9,  $I(q) > 0$ . As the implicit function  $I$  is continuous, there is a point  $y$  on  $pq$  at which  $I(y) = 0$  and  $d(y, p) \leq \tau$ . Since  $x$  is the closest point to  $p$  in  $U$ ,  $d(x, p) \leq d(y, p) \leq \tau$ .  $\square$

**5.2 The  $\tau$ -neighborhood.** To guarantee that  $U$  is a manifold, we have to prove that the gradient of  $I$  is non-zero at each point in  $U$ . We know from the results in Section 5.1 that  $U$  is inside the  $\tau$ -neighborhood of  $F$ . In this section we will study the properties of  $\nabla I(x)$  for points  $x$  inside the  $\tau$ -neighborhood.



For a point  $x$  inside the  $\tau$ -neighborhood,  $B_2(x)$  is defined as a ball of radius  $\sqrt{(|\phi(x)| + \epsilon)^2 + 25\epsilon^2}$  centered at  $x$ . With this new definition of  $B_2(x)$ , it is easy to show that the samples inside  $B_2(x)$  are contained in a small ball centered at the point closest to  $x$  in  $F$ .

**Observation 12.** *Let  $x$  be a point that is inside the  $\tau$ -neighborhood as shown in Figure 6. Let  $p$  be the closest point to  $x$  in  $F$ . All samples inside  $B_2(x)$  are contained in a ball of radius  $6\epsilon$  centered at  $p$ .*

Because of the Lipschitz condition on the surface normals of  $F$ , the difference between the point functions of the samples inside a small ball at a point on the surface is bounded.

**Lemma 13.** *Consider a point  $x$  whose closest point on the surface  $F$  is  $p$ . Let  $\vec{n}$  be the surface normal at  $p$  and let  $B$  be a ball of radius  $6\epsilon$  at  $p$ . For each sample  $s_i \in B$ ,  $P_{s_i}(x) = \phi(x) + \zeta_i$  where  $|\zeta_i| \leq 56\epsilon^2 + 36|\phi(x)|\epsilon^2$ .*

*Proof.* Let  $p_i$  be the closest point to  $s_i$  on the surface. As  $d(p, p_i) \leq 6\epsilon + \epsilon^2$ , the angle between the normal at  $p_i$  and  $\vec{n}$  is less than  $\frac{6\epsilon + \epsilon^2}{1 - 3(6\epsilon + \epsilon^2)}$  from Theorem 1. Let  $\vec{n}_i$  be the normal associated with  $s_i$ . From the sampling conditions the angle between the normal of  $p_i$  and  $\vec{n}_i$  is at most  $\epsilon$ . So the angle between  $\vec{n}_i$  and  $\vec{n}$  is given by,  $\theta < \frac{6\epsilon + \epsilon^2}{1 - 3(6\epsilon + \epsilon^2)} + \epsilon$ . We can now write  $\vec{n}_i = \vec{n} + \vec{\delta}_i$ , where  $\|\vec{\delta}_i\| \leq \frac{\theta}{\sqrt{2}}$ .

$$\begin{aligned} (x - s_i) \cdot \vec{n}_i &= ((x - p) + (p - s_i)) \cdot \vec{n}_i \\ &= (x - p) \cdot \vec{n}_i + (p - s_i) \cdot (\vec{n} + \vec{\delta}_i) \\ &= (x - p) \cdot \vec{n} - (x - p) \cdot (\vec{n} - \vec{n}_i) \\ &\quad + (p - s_i) \cdot (\vec{n} + \vec{\delta}_i). \end{aligned}$$

Because  $p$  is the closest point to  $x$  on the surface,  $(x - p) \cdot \vec{n} = \phi(x)$  and  $(x - p)$  is parallel to  $\vec{n}$ . Since the angle between  $\vec{n}$  and  $\vec{n}_i$  is less than  $\theta$ ,

$$|(x - p) \cdot (\vec{n} - \vec{n}_i)| \leq |\phi(x)|(1 - \cos \theta) \leq \frac{|\phi(x)|\theta^2}{2}.$$

Since sample  $s_i$  is inside  $B$ ,

$$|(p - s_i) \cdot \vec{\delta}_i| \leq (6\epsilon) \frac{\theta}{\sqrt{2}} < 36\epsilon^2.$$

From Observation 2, the distance from each sample inside  $B$  to the tangent plane at  $p$  is at most  $\frac{(6\epsilon + \epsilon^2)^2}{2} + \epsilon^2 < 20\epsilon^2$ . Hence  $(p - s_i) \cdot \vec{n} < 20\epsilon^2$ . We can now write  $P_{s_i}(x) = \phi(x) + \zeta_i$ , where  $|\zeta_i| \leq 56\epsilon^2 + 36|\phi(x)|\epsilon^2$ .  $\square$

We now show that the value of  $I(x)$  is mostly determined by the points inside  $B_2(x)$  by proving results similar to Lemma 7 and Lemma 8 for a point  $x$  inside the  $\tau$ -neighborhood.

**Lemma 14.** *For a point  $x$  in the  $\tau$ -neighborhood,*

$$\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} < c_2.$$

*Proof.* For a point  $x$  inside the  $\tau$ -neighborhood,  $w_0 = \sqrt{(|\phi(x)| + \epsilon)^2 + 25\epsilon^2}$ . Substituting this into Equation 3,

$$\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} \leq \frac{3200}{\epsilon^2} (w_0^2 + \epsilon w_0 + \epsilon^2) e^{-25}.$$

$\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)}$  has the largest value when  $\phi(x) = \tau$  and  $w_0 = \sqrt{34}\epsilon < 6\epsilon$ . So

$$\frac{D_{\text{out}}(x)}{D_{\text{in}}(x)} < \frac{3200}{\epsilon^2} (43\epsilon^2) e^{-25} < c_2. \quad \square$$

Lemma 8 is not true for  $x$  inside the  $\tau$ -neighborhood, as  $|N_{\text{in}}(x)|$  might be zero. However, we can prove an upper bound on  $\frac{N_{\text{out}}(x)}{D(x)}$ . The proof of this Lemma is exactly like the proof of Lemma 14.

**Lemma 15.** *For a point  $x$  inside the  $\tau$ -neighborhood,  $\frac{|N_{\text{out}}(x)|}{D(x)} < c_2\epsilon$ .*

We begin by splitting the contributions to  $\nabla I(x)$  in the following way:  $\nabla I(x) = \nabla I_{\text{in}}(x) + \nabla I_{\text{out}}(x)$ . Here,

$$\nabla I_{\text{in}}(x) = \frac{D_{\text{in}}(x) \nabla N_{\text{in}}(x) - N_{\text{in}}(x) \nabla D_{\text{in}}(x)}{D^2(x)}, \quad (6)$$

and

$$\begin{aligned} \nabla I_{\text{out}}(x) &= \frac{D_{\text{out}}(x) \nabla N_{\text{in}}(x)}{D^2(x)} + \frac{\nabla N_{\text{out}}(x)}{D(x)} \\ &\quad - \frac{N_{\text{out}}(x) \nabla D_{\text{in}}(x)}{D^2(x)} - \frac{N_{\text{in}}(x) \nabla D_{\text{out}}(x)}{D^2(x)} \\ &\quad - \frac{N_{\text{out}}(x) \nabla D_{\text{out}}(x)}{D^2(x)}. \end{aligned} \quad (7)$$

To show that the gradient of  $I$  is never zero inside the  $\tau$ -neighborhood, we will prove a stronger result. For a point  $x$  in the  $\tau$ -neighborhood, we show that  $\nabla I(x) \cdot \vec{n} > 0$  where  $\vec{n}$  is the normal of the point closest to  $x$  on the surface. In Section 6 we will use this result to show that  $U$  is homeomorphic to  $F$ .

The norm of the gradient of Gaussian weight functions decreases exponentially with distance. Using the proof technique of Lemma 8, we can obtain an upper bound on  $\|\nabla N_{\text{out}}\|$  and  $\|\nabla D_{\text{out}}\|$ . Observation 16 summarizes the results.

**Observation 16.** *For a point  $x$  in the  $\tau$ -neighborhood.*

1.  $\left\| \frac{\nabla D_{\text{out}}(x)}{D(x)} \right\| < \frac{c_2}{\epsilon}$ , and
2.  $\left\| \frac{\nabla N_{\text{out}}(x)}{D(x)} \right\| < c_2$ .

We now show that points outside  $B_2(x)$  have little effect on  $\nabla I(x)$  by proving an upper bound on  $\|\nabla I_{\text{out}}(x)\|$ .

**Lemma 17.** *For a point  $x$  in the  $\tau$ -neighborhood,  $\|\nabla I_{\text{out}}(x)\| < c_1$ .*

*Proof.* We will compute the norm of each term in the expression for  $\nabla I_{\text{out}}(x)$  given by Equation 7.

- $\left\| \frac{D_{\text{out}}(x) \nabla N_{\text{in}}(x)}{D^2(x)} \right\|$ .

We can write  $\frac{\nabla N_{\text{in}}(x)}{D(x)}$  as

$$\frac{\nabla N_{\text{in}}(x)}{D(x)} = \frac{\sum_{s_i \in B_2(x)} W_{s_i}(x) (n_i - \frac{2P_{s_i}(x)}{\epsilon^2} (x - s_i))}{\sum_{s_i} W_{s_i}(x)}.$$

Clearly,

$$\left\| \frac{\nabla N_{\text{in}}(x)}{D(x)} \right\| \leq \max_i \left\{ \|n_i\| + \frac{|2P_{s_i}(x)|}{\epsilon^2} \|x - s_i\| \right\}.$$

As  $x$  is inside the  $\tau$ -neighborhood and  $s_i$  is inside  $B_2(x)$ , the point function  $|P_{s_i}(x)| \leq \|x - s_i\| < 6\epsilon$ . So  $\left\| \frac{\nabla N_{\text{in}}(x)}{D(x)} \right\| < 73$ . From Lemma 14,  $\frac{D_{\text{out}}(x)}{D(x)} < c_2$ .

$$\left\| \frac{D_{\text{out}}(x) \nabla N_{\text{in}}(x)}{D^2(x)} \right\| < 73c_2. \quad (8)$$

- $\left\| \frac{\nabla N_{\text{out}}(x)}{D(x)} \right\|$

From Observation 16,

$$\left\| \frac{\nabla N_{\text{out}}(x)}{D(x)} \right\| < c_2. \quad (9)$$

- $\left\| \frac{N_{\text{out}}(x) \nabla D_{\text{in}}(x)}{D^2(x)} \right\|$

$\frac{\nabla D_{\text{in}}(x)}{D(x)}$  can be written as

$$\frac{\nabla D_{\text{in}}(x)}{D(x)} = \frac{\sum_{s_i \in B_2(x)} W_i(x) (-\frac{2}{\epsilon^2} (x - s_i))}{\sum_{s_i} e^{-\|x - s_i\|^2 / \epsilon^2}}.$$

As  $\|x - s_i\| \leq 6\epsilon$ ,  $\left\| \frac{\nabla D_{\text{in}}(x)}{D(x)} \right\| \leq \frac{12}{\epsilon}$ . From Lemma 15,  $\frac{N_{\text{out}}(x)}{D(x)} < c_2\epsilon$ .

$$\left\| \frac{N_{\text{out}}(x) \nabla D_{\text{in}}(x)}{D^2(x)} \right\| < 12c_2. \quad (10)$$

- $\left\| \frac{N_{\text{in}}(x) \nabla D_{\text{out}}(x)}{D^2(x)} \right\|$

For any point  $x$  inside the  $\tau$ -neighborhood,  $\left| \frac{N_{\text{in}}(x)}{D(x)} \right| \leq \max\{P_{s_i}(x) | s_i \in B_2(x)\} \leq 6\epsilon$ . From Observation 16,  $\left\| \frac{\nabla D_{\text{out}}(x)}{D(x)} \right\| < \frac{c_2}{\epsilon}$ .

$$\left\| \frac{N_{\text{in}}(x) \nabla D_{\text{out}}(x)}{D^2(x)} \right\| < 6c_2. \quad (11)$$

- $\left\| \frac{N_{\text{out}}(x) \nabla D_{\text{out}}(x)}{D^2(x)} \right\|$

From Observation 16,  $\left\| \frac{\nabla D_{\text{out}}(x)}{D(x)} \right\| < \frac{c_2}{\epsilon}$ . Combining this with the bound in Lemma 15,

$$\left\| \frac{N_{\text{out}}(x) \nabla D_{\text{out}}(x)}{D^2(x)} \right\| < \frac{c_2}{\epsilon} (c_2\epsilon) < c_2^2. \quad (12)$$

Adding the norms of each term in the expression for  $\nabla I_{\text{out}}(x)$  we get

$$\|\nabla I_{\text{out}}(x)\| < 93c_2 < c_1. \quad \square$$

**Theorem 18.** *Let  $x$  be a point in the  $\tau$ -neighborhood of  $F$  and let  $p$  be the point on  $F$  closest to  $x$ . Let  $\vec{n}$  be the normal of  $p$ . Then for values of  $\epsilon \leq 0.01$ ,  $\vec{n} \cdot \nabla I(x) > 0$ .*

*Proof.* From Lemma 17,

$$\vec{n} \cdot \nabla I_{\text{out}}(x) \leq \|\nabla I_{\text{out}}(x)\| < c_1.$$

We now consider the expression for  $\nabla I_{\text{in}}(x)$ .

$$\begin{aligned} \nabla I_{\text{in}}(x) &= \frac{1}{D^2(x)} \sum_{s_i} \sum_{s_j} W_i(x) W_j(x) \{ \vec{n}_i \\ &\quad + \frac{2P_{s_i}(x)}{\epsilon^2} (s_i - s_j) \}. \end{aligned}$$

The summation is over all samples  $s_i, s_j \in B_2(x)$ . From Observation 12, all samples in  $B_2(x)$  are contained inside a ball of radius  $6\epsilon$  centered at  $p$ . So we can bound the difference between the point functions of samples inside  $B_2(x)$  using Lemma 13.

The point function of each sample  $s_i \in B_2(x)$  can be written as  $P_{s_i}(x) = \phi(x) + \zeta_i$  where  $|\zeta_i| < 56\epsilon^2 + 36\tau\epsilon^2 < 60\epsilon^2$ . Let  $C_{ij}(x) = W_i(x) W_j(x)$ .

$$\begin{aligned} \nabla I_{\text{in}}(x) &= \frac{1}{D^2(x)} \sum_{s_i} \sum_{s_j} C_{ij}(x) (\vec{n}_i + \frac{2|\phi(x)|}{\epsilon^2} (s_i - s_j) \\ &\quad + \frac{2\zeta_i}{\epsilon^2} (s_i - s_j)) \\ &= \frac{1}{D^2(x)} \sum_{s_i} \sum_{s_j} C_{ij}(x) (\vec{n}_i + \frac{2\zeta_i}{\epsilon^2} (s_i - s_j)). \end{aligned}$$

From the above equation,

$$\vec{n} \cdot \nabla I_{\text{in}}(x) \geq \frac{\sum_{s_i} \sum_{s_j} C_{ij}(x)}{D^2(x)} \min_{ij} \{ \vec{n} \cdot (\vec{n}_i + \frac{2\zeta_i}{\epsilon^2} (s_i - s_j)) \}.$$

As we are summing over all samples inside  $B_2(x)$ ,  $\sum_i \sum_j C_{ij}(x) = D_{\text{in}}^2(x)$ . From Lemma 14,  $\frac{D_{\text{in}}^2(x)}{D^2(x)} \geq (1 - c_1)^2$ . Note that the samples inside  $B_2(x)$  are in a ball of radius  $6\epsilon$  centered at  $p$ . Hence we can use Theorem 1 and Observation 2 to obtain an upper bound on the terms that appear in the above equation.

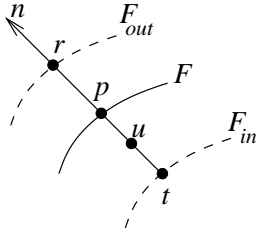


Figure 7: Points  $r, t$  are the closest points to  $p$  on the offset surfaces. The line segment  $rt$  intersects the zero set  $U$  at a unique point  $u$ .

From Theorem 1, the angle between  $\vec{n}$  and  $\vec{n}_i$  is less than  $10\epsilon$ . Hence  $\vec{n} \cdot \vec{n}_i > \cos 10\epsilon$ . From Observation 2, the distance from all samples inside  $B_2(x)$  to the tangent plane at  $p$  is at most  $\frac{(6\epsilon + \epsilon^2)^2}{2} + \epsilon^2$ . Hence  $\vec{n} \cdot (s_i - s_j) \leq 2(\frac{(6\epsilon + \epsilon^2)^2}{2} + \epsilon^2) < 40\epsilon^2$ .

$$\begin{aligned} \vec{n} \cdot \nabla I(x) &\geq \vec{n} \cdot \nabla I_{\text{in}}(x) - \|\nabla I_{\text{out}}(x)\| \\ &\geq (1 - c_1)^2(\cos(10\epsilon) - \\ &\quad \max\{\frac{2\zeta_i}{\epsilon^2} |\vec{n} \cdot (s_i - s_j)|\}) - c_1 \\ &\geq (1 - c_1)^2(\cos(10\epsilon) - 4800\epsilon^2) - c_1. \end{aligned}$$

It is easy to verify that  $\vec{n} \cdot \nabla I(x) > 0$  for values of  $\epsilon \leq 0.01$ .  $\square$

Theorem 18 also proves that the gradient can never be zero inside the  $\tau$ -neighborhood. From Theorem 9, the zero set of  $I$  is inside the  $\tau$ -neighborhood of  $F$ . Hence from the implicit function theorem [7], zero is a *regular* value of  $I$  and the zero set  $U$  is a compact, two-dimensional manifold.

The normal of the reconstructed surface at a point  $u \in U$  is determined by the gradient of the implicit function at  $u$ ,  $\vec{n}_u = \frac{\nabla I(u)}{\|\nabla I(u)\|}$ . Using Theorem 18, we can bound the angle between  $\vec{n}_u$  and the normal of the point closest to  $u$  in  $F$ .

**Theorem 19.** *Let  $u$  be a point on the reconstructed surface  $U$  whose closest point on  $F$  is  $p$ . Let  $\vec{n}_u$  be the normal of  $U$  at point  $u$  and let  $\vec{n}$  be the normal of  $F$  at point  $p$ . An upper bound on the angle  $\theta$  between  $\vec{n}_u$  and  $\vec{n}$  is given by,*

$$\cos \theta \geq \frac{(1 - c_1)^2(\cos(10\epsilon) - 4800\epsilon^2) - c_1}{1 + 2400\epsilon + c_1}.$$

## 6 Topological Properties

We now use the results in Section 5 to define a homeomorphism between  $F$  and  $U$ . As  $F$  and  $U$  are compact, a one-to-one, onto, and continuous function from  $U$  to  $F$  defines a homeomorphism.

**Definition:** Let  $\Gamma : \mathbb{R}^3 \rightarrow F$  map each point  $q \in \mathbb{R}^3$  to the closest point of  $F$ .

**Theorem 20.** *The restriction of  $\Gamma$  to  $U$  is a homeomorphism from  $U$  to  $F$ .*

*Proof.* The discontinuities of  $\Gamma$  are the points on the medial axis of  $F$ . As  $U$  is constrained to be inside the  $\tau$ -neighborhood of  $F$ , the restriction of  $\Gamma$  to  $U$  is continuous.

Now we show that  $\Gamma$  is one-to-one. Let  $p$  be a point on  $F$  and let  $\vec{n}$  be the normal at  $p$  as shown in Figure 7. Consider the line segment parallel to  $\vec{n}$  from  $r$  to  $t$ . At each point  $y \in rt$ ,  $\nabla I(y) \cdot \vec{n} > 0$  from Theorem 18. So the function  $I(x)$  is monotonically decreasing from  $r$  to  $t$  and there is a unique point  $u$  on  $rt$  where  $I(u) = 0$ . Assume there is another point  $v \in U$  for which  $\Gamma(v) = x$ . The point  $v$  has to be outside the segment  $rt$  and the distance from  $v$  to its closest point on  $F$  is greater than  $\tau$ . This contradicts Theorem 10.

Finally we need to show that  $\Gamma$  is onto. As  $\Gamma$  maps closed components of  $U$  onto closed components of  $F$  in a continuous manner,  $\Gamma(U)$  should consist of a set of closed connected components. Consider the point  $p$  in Figure 7. Assume that  $q = \Gamma(u)$  is not in the same component of  $F$  as  $p$ . Let  $B_u$  be the ball of radius  $\tau$  centered at  $u$  that intersects two components of  $F$ , one containing point  $p$  and one containing point  $q$ . Boissonnat and Cazals [9] (Proposition 12) show that any ball whose intersection with  $F$  is not a topological disc contains a point of the medial axis of  $F$ . Since point  $p$  is inside the ball  $B_u$  that contains a point of the medial axis,  $\text{lfs}(p) \leq 2\tau$ . Recall that  $\tau = 2\epsilon$  and that our sampling conditions require  $\epsilon \leq 0.01$ . Hence,  $\text{lfs}(p) \leq 2\tau \leq 0.04$ . This violates our assumption that  $\text{lfs}(p) \geq 1$ .  $\square$

## 7 Discussion

One disadvantage of our algorithm is that it requires sample normals. However, approximate sample normals can be easily obtained for laser range data by triangulating the range images. Each sample normal can be oriented using the location of the range scanner. When oriented normals are unavailable, the absolute distance to the tangent plane at each sample can be used instead of the signed distance as a point function to define a new function  $I_u(x)$ . The zero set of this function is hard to analyze as its gradient is not smooth near the sample points. However, the results in this paper can be easily extended to show that the  $\tau$ -level set of  $I_u(x)$  consists of two components on each side of the surface, each homeomorphic to  $F$ .

Our sampling requirements are determined by the smallest local feature size of a point on  $F$ . Recall that the width of the Gaussian functions used in our algorithm depends on the smallest local feature size. When sampling density is proportional to the local feature size, the width of the Gaussian weight functions might be much smaller than the spacing between sample points in areas of the surface with large local feature size. As a result, the reconstructed surface will be noisy and might have the wrong topology. One solution is to make the width of the Gaussian weight functions

proportional to the spacing between sample points. We are currently working on extending the MLS construction and our proofs to deal with adaptively sampled surfaces.

The implicit surface we construct in this paper only passes near the sample points, but we can construct a surface that interpolates the sample points with weight functions such as  $W_s(x) = \frac{e^{-\|x-s\|^2}}{\|x-s\|^2}$ , that are infinite at the sample points. We can prove that the zero set is restricted to the  $\tau$ -neighborhood when this weight function is used, but, we could not prove results about the gradient approximations.

## Acknowledgments

I would like to thank Nina Amenta, James O'Brien and my advisor Jonathan Shewchuk for helpful comments. I would also like to thank François Labelle for reading the proofs and for pointing out an error in an earlier version of the paper.

## References

- [1] A. ADAMSON AND M. ALEXA, *Approximating and Intersecting Surfaces from Points*, in Proceedings of the Eurographics Symposium on Geometry Processing, Eurographics Association, 2003, pp. 230–239.
- [2] M. ALEXA, J. BEHR, D. COHEN-OR, S. FLEISHMAN, D. LEVIN, AND C. T. SILVA, *Computing and Rendering Point Set Surfaces*, IEEE Transactions on Visualization and Computer Graphics, 9 (2003), pp. 3–15.
- [3] N. AMENTA AND M. BERN, *Surface Reconstruction by Voronoi Filtering*, Discrete & Computational Geometry, 22 (1999), pp. 481–504.
- [4] N. AMENTA, S. CHOI, T. K. DEY, AND N. LEEKHA, *A Simple Algorithm for Homeomorphic Surface Reconstruction*, International Journal of Computational Geometry and Applications, 12 (2002), pp. 125–141.
- [5] N. AMENTA, S. CHOI, AND R. KOLLURI, *The Power Crust*, in Proceedings of the Sixth Symposium on Solid Modeling, Association for Computing Machinery, 2001, pp. 249–260.
- [6] N. AMENTA AND Y. KIL, *Defining Point-Set Surfaces*, ACM Transactions on Graphics, 23 (2004), pp. 264–270.
- [7] J. BLOOMENTHAL, ed., *Introduction to Implicit Surfaces*, Morgan Kaufman, 1997.
- [8] J.-D. BOISSONNAT AND F. CAZALS, *Smooth Surface Reconstruction via Natural Neighbour Interpolation of Distance Functions*, in Proceedings of the Sixteenth Annual Symposium on Computational geometry, ACM, 2000, pp. 223–232.
- [9] ———, *Natural Neighbor Coordinates of Points on a Surface*, Computational Geometry Theory and Applications, 19 (2001), pp. 155–173.
- [10] J.-D. BOISSONNAT, D. COHEN-STEINER, AND G. VEGTER, *Isotopic Implicit Surface Meshing*, in Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 2004, pp. 301–309.
- [11] J. D. BOISSONNAT AND S. OUDOT, *Provably Good Surface Sampling and Approximation*, in Proceedings of the Eurographics Symposium on Geometry Processing, Eurographics Association, 2003, pp. 9–18.
- [12] J. C. CARR, R. K. BEATSON, J. B. CHERRIE, T. J. MITCHELL, W. R. FRIGHT, B. C. MCCALLUM, AND T. R. EVANS, *Reconstruction and Representation of 3D Objects with Radial Basis Functions*, in Computer Graphics (SIGGRAPH 2001 Proceedings), Aug. 2001, pp. 67–76.
- [13] B. CURLESS AND M. LEVOY, *A Volumetric Method for Building Complex Models from Range Images*, in Computer Graphics (SIGGRAPH '96 Proceedings), 1996, pp. 303–312.
- [14] T. K. DEY AND S. GOSWAMI, *Provable Surface Reconstruction from Noisy Samples*, in Proceedings of the Twentieth Annual Symposium on Computational Geometry, Brooklyn, New York, June 2004, Association for Computing Machinery.
- [15] S. FLEISHMAN, M. ALEXA, D. COHEN-OR, AND C. T. SILVA, *Progressive Point Set Surfaces*, ACM Transactions on Computer Graphics, 22 (2003).
- [16] H. HOPPE, T. DEROSE, T. DUCHAMP, J. McDONALD, AND W. STUETZLE, *Surface Reconstruction from Unorganized Points*, in Computer Graphics (SIGGRAPH '92 Proceedings), 1992, pp. 71–78.
- [17] D. LEVIN, *Mesh-Independent Surface Interpolation*, in Geometric Modeling for Scientific Visualization, G. Brunett, B. Hamann, K. Mueller, and L. Linsen, eds., Springer-Verlag, 2003.
- [18] M. LEVOY, K. PULLI, B. CURLESS, S. RUSINKIEWICZ, D. KOLLER, L. PEREIRA, M. GINTON, S. ANDERSON, J. DAVIS, J. GINSBERG, J. SHADE, AND D. FULK, *The Digital Michelangelo Project: 3D Scanning of Large Statues*, in Computer Graphics (SIGGRAPH 2000 Proceedings), 2000, pp. 131–144.
- [19] W. E. LORESEN AND H. E. CLINE, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, in Computer Graphics (SIGGRAPH '87 Proceedings), July 1987, pp. 163–170.
- [20] N. J. MITRA, N. GELFAND, H. POTTMANN, AND L. GUIBAS, *Registration of Point Cloud Data from a Geometric Optimization Perspective*, in Symposium on Geometry Processing, 2004.
- [21] N. J. MITRA, A. NGUYEN, AND L. GUIBAS, *Estimating Surface Normals in Noisy Point Cloud Data*, International Journal of Computational Geometry and Applications, 14 (2004), pp. 261–276.
- [22] Y. OHTAKE, A. BELYAEV, M. ALEXA, G. TURK, AND H.-P. SEIDEL, *Multi-Level Partition of Unity Implicit*, ACM Transactions on Graphics, 22 (2003), pp. 463–470.
- [23] S. OSHER AND R. FEDKIW, *The Level Set Method and Dynamic Implicit Surfaces*, Springer-Verlag, New York, 2003.
- [24] M. PAULY, R. KEISER, L. P. KOBELT, AND M. GROSS, *Shape Modeling with Point-Sampled Geometry*, ACM Trans. Graph., 22 (2003), pp. 641–650.
- [25] C. SHEN, J. F. O'BRIEN, AND J. R. SHEWCHUK, *Interpolating and Approximating Implicit Surfaces from Polygon Soup*, ACM Transactions on Graphics, 23 (2004), pp. 896–904.
- [26] G. TURK AND J. O'BRIEN, *Shape Transformation Using Variational Implicit Functions*, in Computer Graphics (SIGGRAPH '99 Proceedings), 1999, pp. 335–342.
- [27] H.-K. ZHAO, S. OSHER, AND R. FEDKIW, *Fast Surface Reconstruction Using the Level Set Method*, in First IEEE Workshop on Variational and Level Set Methods, 2001, pp. 194–202.



## Medical Applications of Implicit Surfaces

Terry S. Yoo

Office of High Performance Computing and Communications

National Library of Medicine, NIH



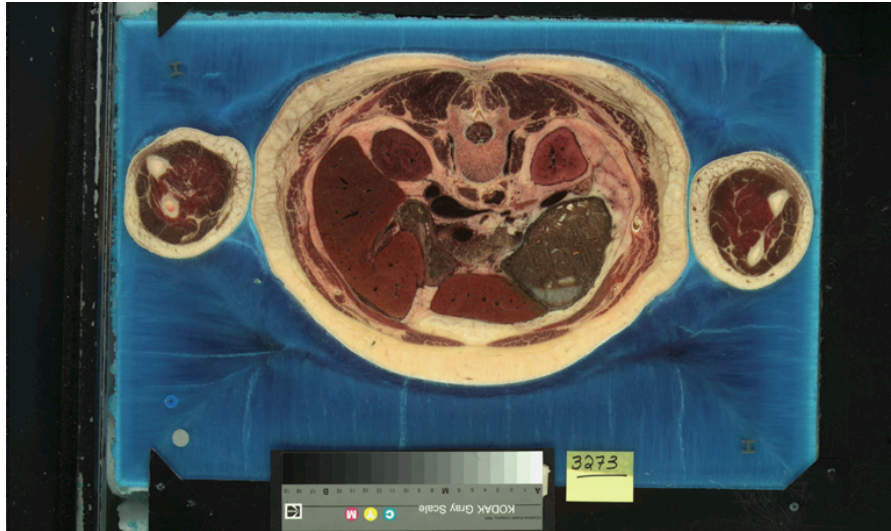
## Acknowledgements

- Volume Modeling Consortium
  - Bryan Morse (BYU)
  - K.R. Subramanian (UNCC)
  - Penny Rheingans (UMBC)
  - Kathleen Hoffman (UMBC)
  - David T. Chen (NLM/NIH)
  - Terry S. Yoo (NLM/NIH)
- Also
  - Greg Turk (GA Tech)
  - James F. O'Brien (UCB)

## Voxel Bitmaps for modeling?



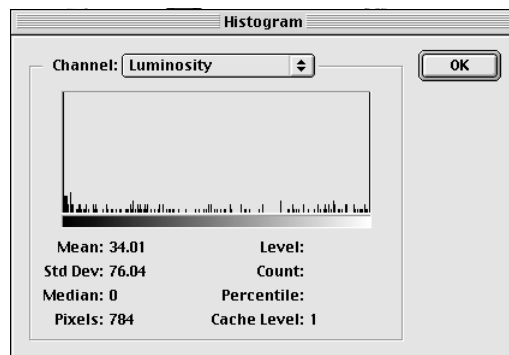
SIGGRAPH2005



## Voxel Bitmaps for modeling?



SIGGRAPH2005



## Philosophical Problem



- Modeling and representing anatomical or other medical objects requires a respect for detail and a commitment to the truth.
- Discrete representations lead to sampling error, aliasing.
- Under magnification, approximations become visible.
- What is an ideal modeling/interpolating primitive?
  - Invariant with respect to rotation, translation, and zoom

## General Problem



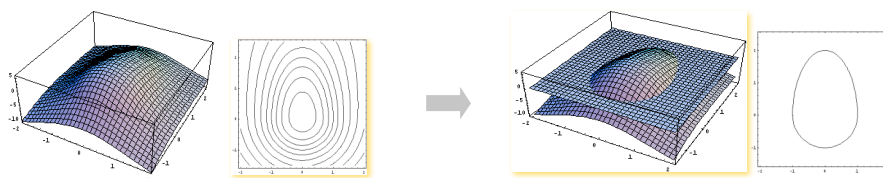
- Medical imaging produces discretely sampled images or volumes.
- May want/need to have continuous surfaces fit to those discrete points for analysis.
- Could a class of interpolants known as interpolating (or constrained) implicit surfaces be used?
- Lots and lots of points! Too slow!
- Or is it?

## Outline



- Summarize methods for constrained implicit surfaces
- Some successful results
- Examine the problem posed by large data
  - Deficiencies of thin-plate splines in medical modeling
- Propose alternative radial basis functions
- Results
- Future directions

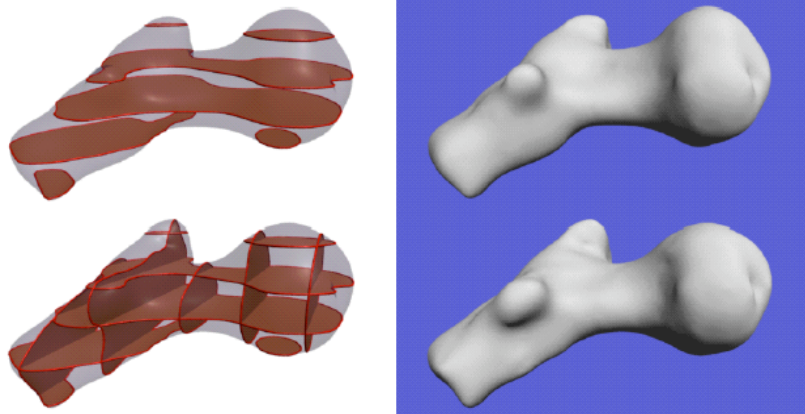
## Implicit Surfaces



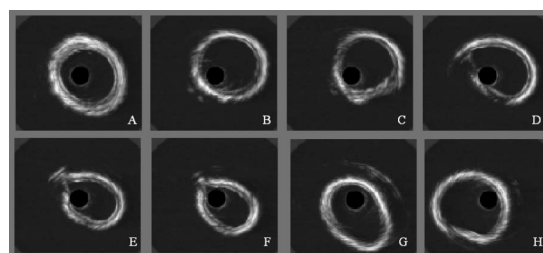
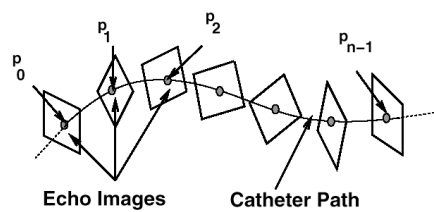
- Build a embedding or characteristic function.
- One connected isosurface is the implicit surface.
- Related to research in level sets.
- Not an parametric or polygonal surface representation.



## Medical Implicit Models (from Turk and O'Brien 1999)



## 3D Reconstructions from Intravenous Ultrasound

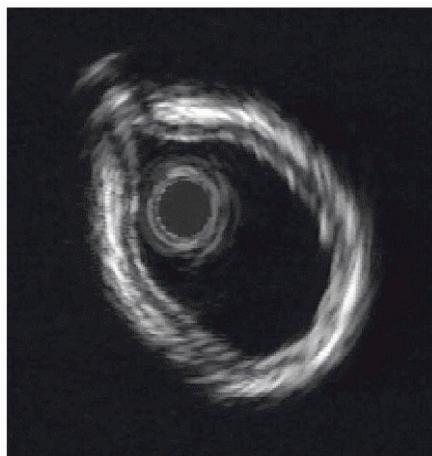


## 3D Ultrasound Reconstruction



- Implicit surfaces using thin plate spline radial basis functions.
- Reconstructs 3D shapes from arbitrarily oriented segmented 2D contours.
- Thin plate splines are useful for interpolating over large, unpredictable distances.
- Requires segmentation (easy for IVUS)

## Polygonal Reconstruction



## Implicit Reconstruction



## Other Implications for Ongoing Work



- Faster method means interpolating implicit surfaces are viable for medical models
  - Resampling models
  - More accurate model simplification
- Local support means local effect
  - Incremental updating
  - Interactive modeling

## Current Work



- Error analysis vs. thin-plate splines
- More efficient data structures for managing spatial locality
- Other types of sparse solvers
  - Using LU, could use SVD, CG, etc.
- Better ways to deal with extracting desired isosurface and not the inner/outer hulls
- More timing studies/profiling

## Current Work: Reinventing Active Surfaces

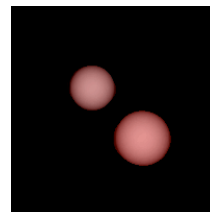
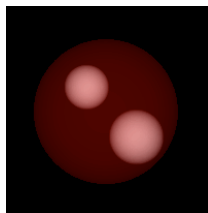
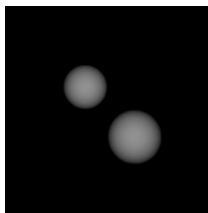


- Implicit surfaces can interpolate 2-D segments (unevenly sampled slices).
- Implicit Interpolating methods might provide better (smoother, more continuous) priors to initialize active contours and level set segmenters.
- Implicit Interpolators can be used to regularize stacks of hand segmented slices (adapted from Yngve & Turk).

## What are Active Surfaces?



- Active Surfaces are algorithms that minimize energy to detect object boundaries.



## Traditional Active Contours



- Combined Bottom-up and Top-down approach
  - Original Image
  - User Interaction

Minimize Energy Function:

$$E_{Contour} = E_{Internal} + E_{External}$$

$E_{Internal}$  — Keeps contour smooth.

$E_{External}$  — Fits the image data.

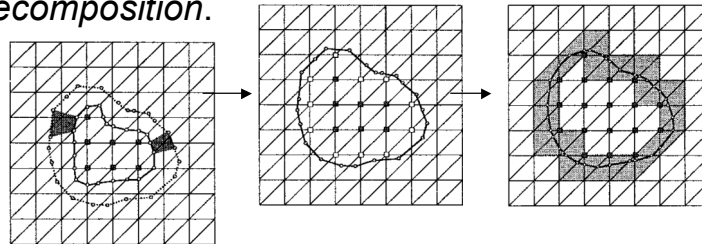


## Previous Work: T-Surfaces



SIGGRAPH2005

- Parametric approach.
- Topology adaptive through *Affine Cell Image Decomposition*.



From *Topology Adaptive Deformable Surfaces for Medical Image Volume Segmentation*

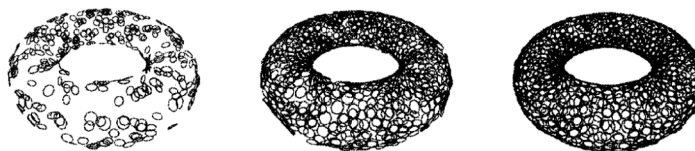


## Previous Work: Dynamic Particles



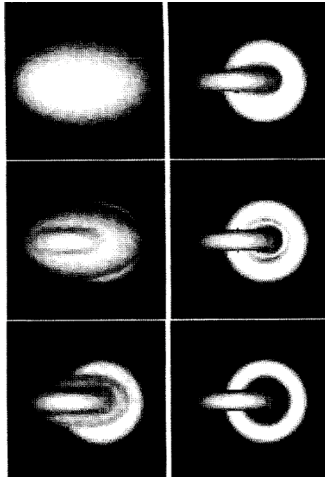
SIGGRAPH2005

1. Particle system discovers topological and geometric surface structure.
2. Efficient Triangulation scheme.



From *Modeling Surfaces of Arbitrary Topology with Dynamic Particles*

## Previous Work: Level-Set



From *Geodesic Active Contours*

- Implicit Representation
- Topology Adaptive for free
- Move based on intrinsic geometry measures of the image.

## Previous Active Surface Work



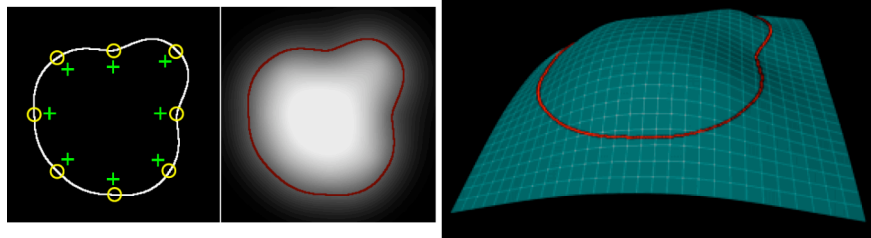
T-Surfaces	Topology Adaptive (with special procedure)	User Interaction
Dynamic Particles	Topology Adaptive (with heuristic)	User Interaction (difficult)
Level-Set	Topology Adaptive	No User Interaction

## Constraint-Based Implicit



SIGGRAPH2005

- Implicit Representation
- Topology Adaptive for free
- Not tied to mesh
- Scattered Data Interpolation



## Constraint-Based Implicit Model



SIGGRAPH2005

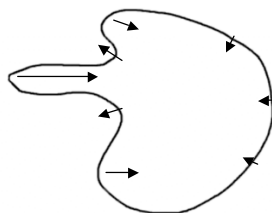
- Image Energy
- Internal Energy
- Balloon Force
- Repulsion Force

$$F_{image} = ((\|\nabla G_{\sigma} * O\|) \cdot N)N$$

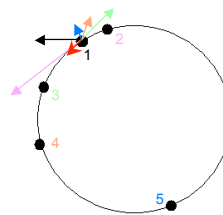
$$F_{mcf} = -K_m N$$

$$F_{balloon}(c_i) = F(I(c_i))N(c_i)$$

$$F_{coulomb}(c_i) = F_{C_i} - (N_i \cdot F_{C_i})N_i$$



Mean Curvature Flow



Coulomb Force

$$F_{C_i} = \sum_{j \neq i} \frac{v_i - v_j}{\|v_i - v_j\|^3}$$



## Three-Dimensional Extensions



- Constraint Addition
  - Gram-Schmidt orthogonalization to create local coordinate system
  - Starting with  $n = (f_x, f_y, f_z)$
  - And  $i = (1,0,0)$  or  $(0,0,1)$  or  $(0,1,0)$
  - Calculate  $u = n \times i$  and  $v = n \times u$
  - Use combinations of u and v for placement of new constraints

## Three-Dimensional Extensions

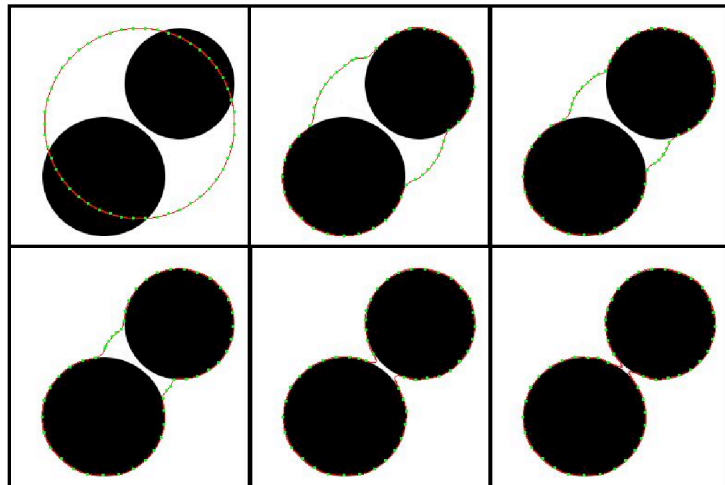


- Curvature
  - Use previously discussed Gram-Schmidt orthogonalization to create local coordinate system u and v.
  - Use  $K_1 = u^T H u$  and  $K_2 = v^T H v$
  - Where H the Hessian is equal to

$$H = \begin{pmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{xy} & f_{yy} & f_{yz} \\ f_{xz} & f_{yz} & f_{zz} \end{pmatrix}$$

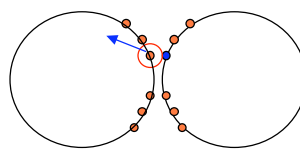
- Calculating mean curvature  $K_m = \frac{(k_1 + k_2)}{2}$

## Modified Coulomb Force Calculation

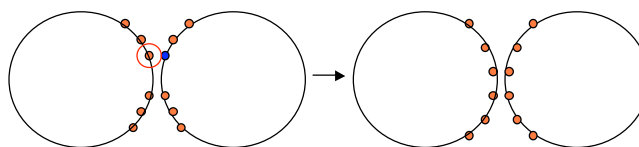


Original constraint-based implicit approach

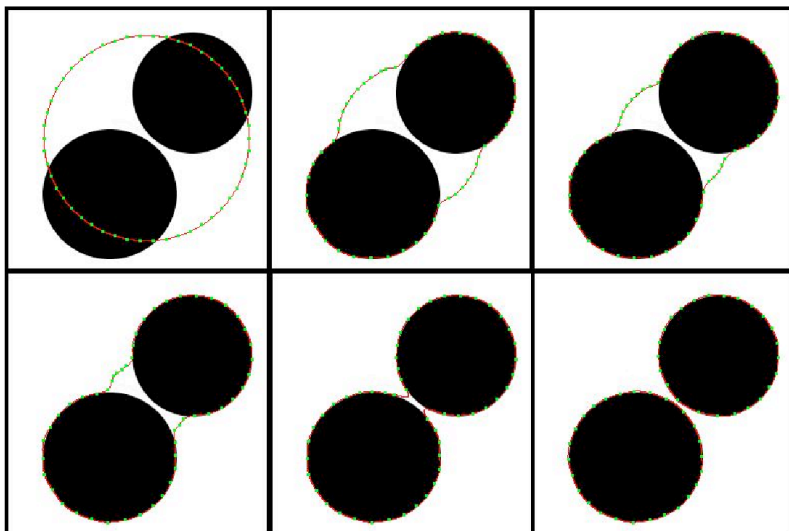
## Modified Coulomb Force Calculation



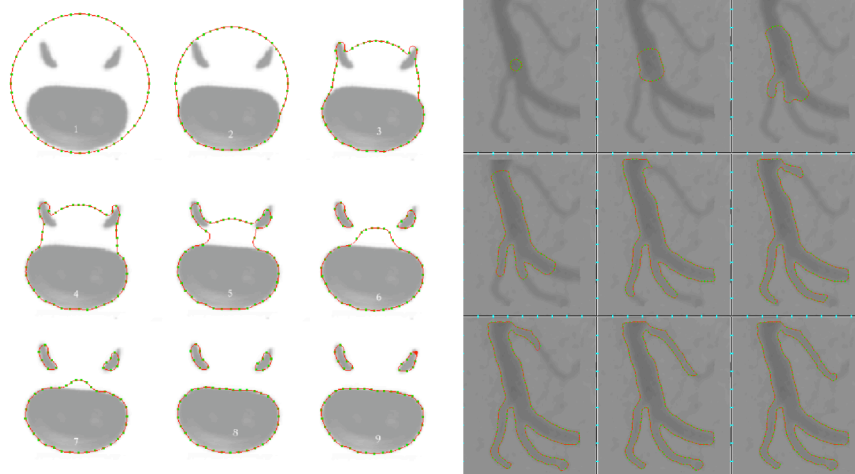
$$F_{C_i} = \sum_{j \neq i} \frac{v_i - v_j}{\|v_i - v_j\|^3} (N_i \cdot N_j)$$



## Modified Coulomb Force Calculation



## Automatic Constraint Addition

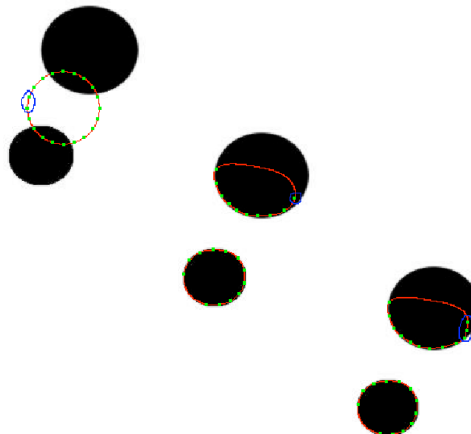


Original constraint-based implicit approaches

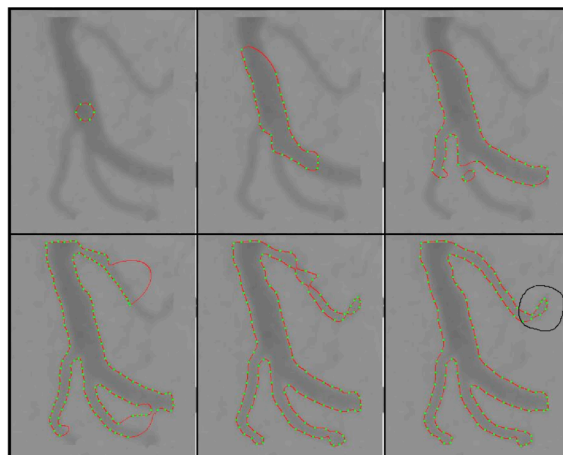
## Automatic Constraint Addition



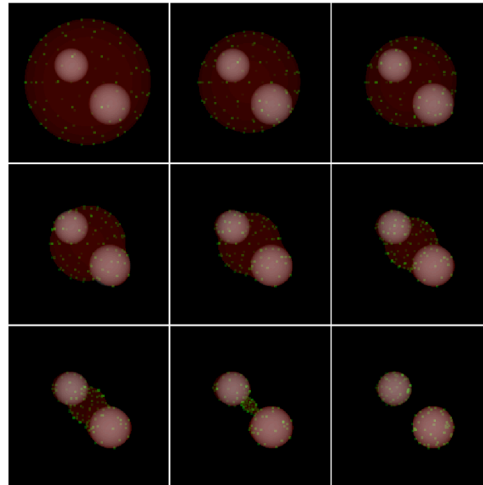
1. Find the initial distance between constraints.
2. After each iteration of evolution check to see if any constraint is further than the desired distance away from any other constraint.
3. If the constraint is "lonely" add four new constraints surrounding it.



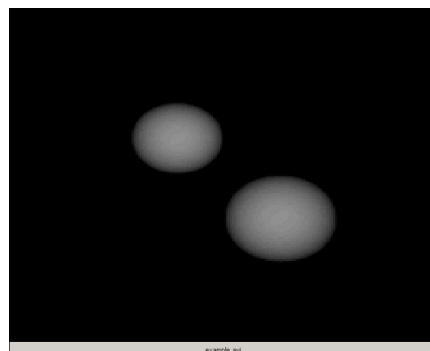
## Automatic Constraint Addition



## Basic Implementation

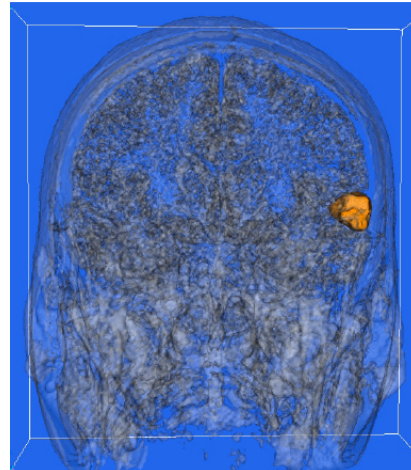
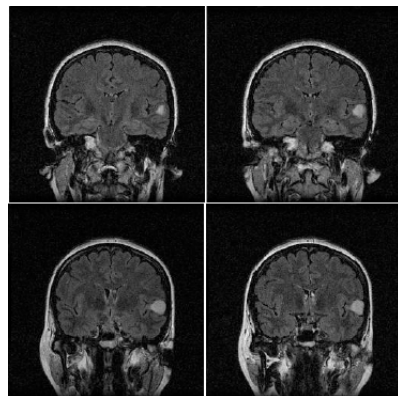


## Result



## Implicit Snakes

Subramanian 2003



## Other Implications for Future Work



- Faster method means interpolating implicit surfaces are viable for medical models
  - Resampling models
  - More accurate model simplification
- Local support means local effect
  - Incremental updating
  - Interactive modeling

## Future Work



- Error analysis vs. thin-plate splines
- More efficient data structures for managing spatial locality
- Other types of sparse solvers
  - Using LU, could use SVD, CG, etc.
- Better ways to deal with extracting desired isosurface and not the inner/outer hulls
- More timing studies/profiling

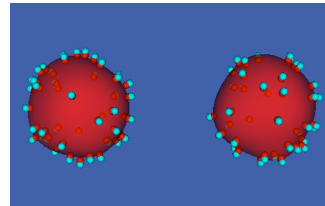
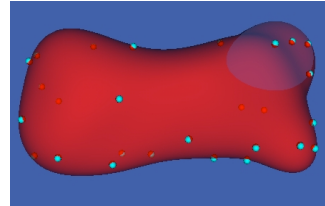
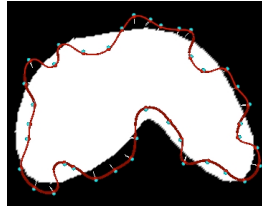
## Future Work: Reinventing 2D segmentation



- Implicit surfaces can interpolate 2-D segments (unevenly sampled slices).
- Implicit Interpolating methods might provide better (smoother, more continuous) priors to initialize active contours and level set segmenters.
- Implicit Interpolators can be used to regularize stacks of hand segmented slices (adapted from Yngve & Turk).

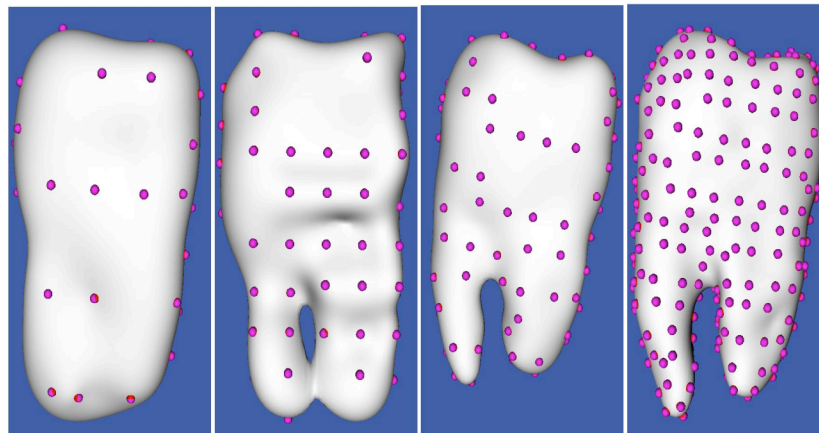
## Implicit Snakes

Yoo and Subramanian 2001



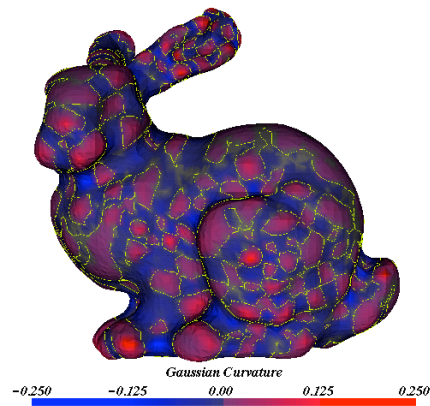
## Implicit Snakes (cont.)

Yoo and Subramanian 2001

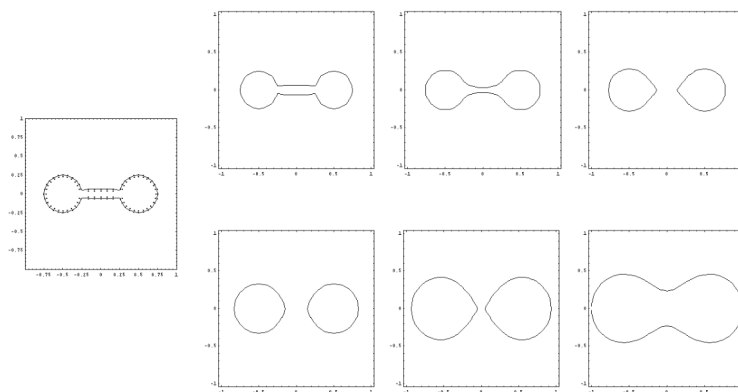




## Future Work: Geometry of Complex Models



## Future: Scale Space Study of Implicit Surfaces



## Conclusions

---



- Implicit functions interpolated with radial basis functions are flexible and easily applied to anatomical modeling.
- Choice of the RBF depends on application (number of points).
- A little computer science goes a long way.
- Leads to interesting interdisciplinary work.
  - computer vision, computational geometry
  - computer graphics, visualization, medicine

# Anatomic Modeling from Unstructured Samples Using Variational Implicit Surfaces

Terry S. Yoo<sup>1</sup>, Bryan Morse<sup>2</sup>, K.R. Subramanian<sup>3</sup>,  
Penny Rheingans<sup>4</sup>, Michael J. Ackerman<sup>1</sup>

<sup>1</sup>*National Library of Medicine, National Institutes of Health, Bethesda, MD 20894 USA*

<sup>2</sup>*Dept. of Computer Science, Brigham Young University, Provo UT 84602 USA*

<sup>3</sup>*Dept. of Computer Science, Univ. of North Carolina Charlotte, Charlotte NC 28223 USA*

<sup>4</sup>*Dept. of CSEE, Univ. Maryland Baltimore County, Baltimore MD 21250 USA*

**Abstract.** We describe the use of variational implicit surfaces (level sets of an embedded generating function modeled using radial basis interpolants) in anatomic modeling. This technique allows the practitioner to employ sparsely and unevenly sampled data to represent complex biological surfaces, including data acquired as a series of non-parallel image slices. The method inherently accommodates interpolation across irregular spans. In addition, shapes with arbitrary topology are easily represented without interpolation or aliasing errors arising from discrete sampling. To demonstrate the medical use of variational implicit surfaces, we present the reconstruction of the inner surfaces of blood vessels from a series of endovascular ultrasound images.

## 1. Introduction

Medical data analysis and visualization often requires the representation of known segments or objects in an easily processed form. The primitives used for these representations are often either a binary voxel map or a polygonal or polyhedral representation. Each of these modeling primitives has its limitations. As a modeling representation, voxel maps are not invariant with respect to rotation or changes in scale. When arbitrarily rotating or scaling a binary voxel map, grey values will be introduced near the boundary as voxels become partially covered by the rotated or scaled bitmap, making necessary a type change from a binary-valued array to a scalar valued system. The alternative is to enforce a binary mapping of the voxel representation, accepting the subsequent sampling errors.

Surface representations modeled either with polygons (or with volume structuring primitives such as tetrahedra) are invariant with respect to rotation and scale. However, if you create a close-up view of the surface representation, the discretization of the surface into planar primitives becomes apparent, and the viewed surface no longer closely approximates the desired smoothness for the representation of the model. Thus although invariant with respect to changes in scale, polygonal surface models are susceptible to artifacts and errors in the piecewise planar approximation to smooth surfaces when scaled.

What is desired is a smooth representation that is easily captured from a binary segmentation and that is invariant with respect to rotation, translation, and scale. Moreover, it should support resampling of the surface model at any arbitrary sampling rate to support visualization at any level of zoom or scale. We have explored the use of variational implicit surfaces as a modeling primitive for binary anatomical objects. These systems provide the necessary smooth differentiable surface models with the desired properties for robust representation of complex biological structures.

## 2. Background and History

When viewing the 3D relationships among anatomical structures identified within medical data, researchers often apply direct volume rendering techniques [6]. The generation of surface models from volumetric information is an alternative to direct rendering and is also a common practice in the visualization of anatomy. Reconstruction of piecewise planar polygonal surface models from rectilinear volume data is frequently achieved using computationally efficient methods such as Marching Cubes [7]. However, when either of these techniques is applied to objects that are represented as binary bitmaps, sampling and discretization errors arise leading to terracing and jaggies, a prominent problem in medical imaging.

Recent work in surface fitting has addressed these issues. Gibson extracts smooth surface models treating the existing binary data as a constraining element in an energy-minimizing deformable surface system [4]. The resulting data structure can be used either to create Euclidean distance maps for direct volume rendering or employed directly as a polygonal surface model [5]. Whitaker has modified the constrained deformable surface model to a constrained level-set model, which creates smooth models while bypassing the need for a separate surface representation [14]. However, while these methods generate smooth representations, both the level set model and the surface net remain discretely sampled and retain the problem that they are not zoom invariant. A different modeling primitive is still needed.

Turk and O'Brien adapt earlier work on thin plate splines [1][2] and radial basis interpolants [3][10] to create a new technique in generating implicit surfaces [11]. Their method allows direct specification of a complex surface from sparse, irregular surface samples. The method is quite flexible and has been extended to higher dimensions to support shape interpolation [12]. However, their technique as described cannot be used to model surfaces where large numbers of surface points are included, making it unsuitable for medical applications where range data or tomographic reconstruction often lead to data described by hundreds of thousands of surface points.

## 3. Methods & Tools

An *implicit surface* is defined by  $\{\bar{x}: f(\bar{x}) = k\}$ ,  $k \in \Re$ , for some characteristic embedding function  $f: \Re^3 \rightarrow \Re$ . Given a set of surface points  $C = \{\bar{c}_1, \bar{c}_2, \bar{c}_3, \dots, \bar{c}_n\}$ , the variational implicit technique interpolates the smoothest possible scalar function  $f(\bar{x})$  whose zero level set,  $\{\bar{x}: f(\bar{x}) = 0\}$ , passes through all points in  $C$ . That is, find the smoothest function  $f$  such that  $f(\bar{x}_i) = 0$  for each known surface point  $\bar{x}_i$ , and  $f(\bar{y}_i) = 1$  for one or more points  $\bar{y}_i$  known to be inside the shape. Following Turk and O'Brien's generalization of the

problem, ., given a set of positions  $\bar{c}_i$  and corresponding values  $h_i$ , solve for an embedding function  $f$  such that  $f(\bar{c}_i) = h_i$ . by employing radial basis interpolating functions  $\phi(r)$  in a critically constrained linear system of  $n$  equations and  $n$  unknowns. Specifically, given a radial basis interpolating function  $\phi(r)$  and a series of known points where the desired function  $f$  is constrained to be  $f(\bar{c}_i) = h_i$ , solve the following equation for the unknown weights  $d_j$ .

$$f(\bar{c}_i) = \sum_{j=1}^n d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i \quad (1)$$

Expanding  $\bar{c}_i = (c_i^x, c_i^y, c_i^z)$ , the entire linear system can be expressed as an  $(n+4) \times (n+4)$  matrix  $M$  where:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2)$$

The resulting matrix is known to be positive semi-definite, and can be solved with a linear decomposition system. Once the weights  $d_i$  are found, the embedding function can be written as:

$$f(\bar{x}) = \sum_{i=1}^n d_i \phi(\|\bar{x} - \bar{c}_i\|) + P(\bar{x}) \quad (3)$$

Depending on the differentiability of the radial basis function,  $\phi(r)$ ,  $f(\bar{x})$  is an implicit surface generating function that can be resampled with arbitrary precision and remains invariant under rotation, translation, and zoom. (For most of this work, we have used the same radial basis function,  $\phi(r) = r^3$  for  $r \geq 0$ , used by Turk and O'Brien from related research on thin-plate splines.) If the constraints  $\bar{c}_i$  for  $f(\bar{x})$  are abstracted from either a grey-level or a binary bit mask volume sampled volume, the resulting variational implicit surface is a more compact analytical representation of the same data.

#### 4. Results

We have applied this method to the modeling of a bovine aorta from data acquired using an endovascular ultrasound transducer. This particular modality acquires noisy 2D image slices, sampled at uneven intervals with non-parallel orientations. The transducer is drawn slowly through the vessel, acquiring cross-sectional sonographic images (see Figure 1).

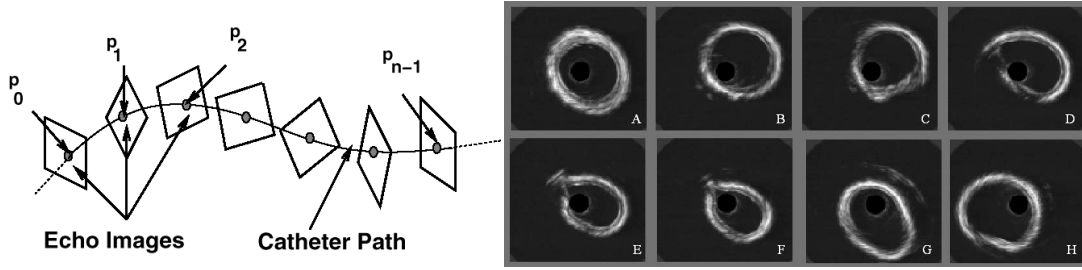


Figure 1. Cross-sectional ultrasound images of an ex-vivo bovine aorta. The slices are acquired at angles perpendicular to the endovascular catheter path. The imaging path is determined by the geometry of the blood vessel.

Each individual slice is then segmented, and the aggregate contours from the tilted slices are processed to form a variational implicit surface. The resulting analytic description can be sampled and rendered using volume rendering approaches, or it can be interrogated and tessellated into a polygonal or parametric surface representation. Figure 2 shows the two views of the interior surface of the bovine aorta from Figure 1 reconstructed as a variational implicit surface. The zero-set of the implicit model has been extracted using a surface tiler, rendering them as polygons at the resolution desired for the magnification shown. If close-up views are desired, the model can easily be re-interrogated and tiled in the surface rendering case or the model simply re-rendered in a direct volume rendering system with arbitrary precision, eliminating aliasing artifacts in the representation of the model.

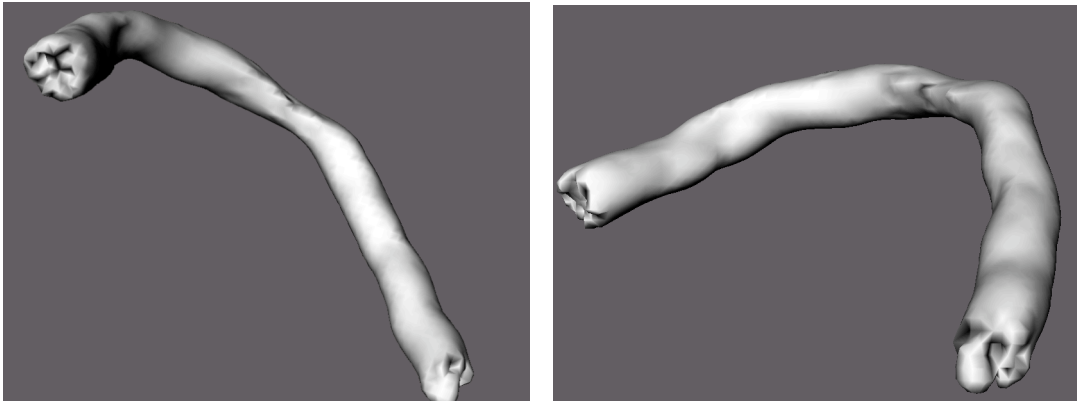


Figure 2. Surface renderings of the interior surfaces of the ex-vivo bovine aorta reconstructed as a variational implicit surface from the ultrasound slices in Figure 1. The models are not subject to aliasing artifacts generated by comparable surface or volume. In addition, these models are generated from sparse, non-uniformly sampled non-parallel ultrasound slices.

## 5. Discussion and Future Work

The radial basis function,  $\phi(r) = r^3$  for  $r \geq 0$ , advocated by Turk and O'Brien is infinite in extent. It has advantages when interpolating across unpredictable spans, making it ideal for morphing and shape interpolation. However, the infinite extent leads to ill-conditioned matrices and increased computational complexity. A naïve approach is easily order  $O(n^3)$ .

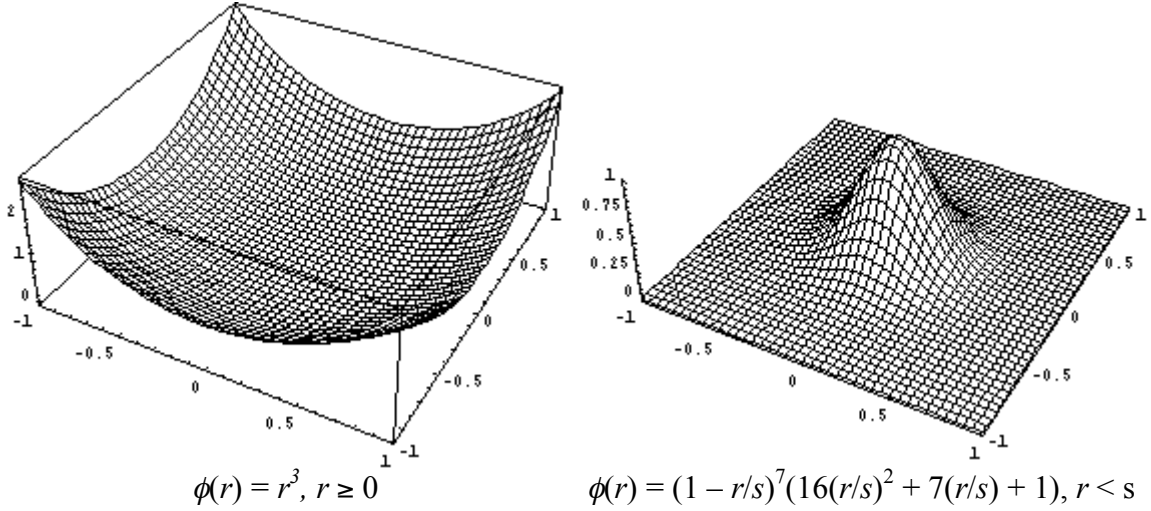


Figure 3. Two radial basis functions. The left function  $\phi(r) = r^3$ ,  $r \geq 0$  has infinite extent. The right function  $\phi(r) = (1 - r/s)^7(16(r/s)^2 + 7(r/s) + 1)$  is clamped to zero outside a specified radius of support  $s$ , and is proven to be  $C^4$  continuous. The compactly supported function on the right leads to more stable numerics and faster solutions of the variational implicit surface model.

In related work, we address the computational complexity of variational implicit surface modeling. Instead of  $\phi(r) = r^3$  as the underlying interpolant, we address select from among a family of radial basis functions presented by Wendland [13] to find a function with the necessary continuity but also with finite, compact local support. Figure 3 shows a comparison of the 3D thin-plate-spline radial basis function and the suggested compactly supported radial basis functions of our current research. The shift from a radial basis function of infinite extent to one that has compact local support has created dramatic gains in memory utilization and computational complexity. Previous work described solutions for systems of equations of order  $O(n^3)$  complexity with iterative solutions capable of achieving order  $O(n^2)$ . The shift to finite interpolants and sparse matrices has shifted the bulk of the computation toward order  $O(n)$ , depending on the complexity of the model and the uniformity of the density of the surface constraints. We have measured the complexity of the matrix solution for some test cases as  $O(n^{1.5})$ . For more details, see Morse [8].

The use of compactly supported radial basis functions imposes a restriction on the maximum distance allowed between systems of surface constraints. This trade-off between speed and the granularity of the surface samples is the topic of some of our future research in this area. The infinite radial basis function permits interpolation across wide spans and with arbitrary orientations to the subsets of constraints that comprise the initial surface description. However, the costs of using this system rise with the number of points required to faithfully represent the surface. This trade-off has ramifications in the choice and precision of the segmentation algorithm to be used and the complexity of the objects to be represented.

Future work on this topic includes the development of hybrid representations that incorporate both infinite and compact radial basis functions. In addition, we will explore advanced slice-based segmentation techniques with confidence in our ability to interpolated smoothly between 2D segments. It should be noted that variational implicit

surfaces can be generated as the output of segmentation systems; however, as a means of smoothly interpolating between sample slices, they can be used to initialize a deformable contour for segmenting an unknown intermediate slice. The use of variational implicit surfaces to help generate priors for segmentation systems is a current focus for some of our group. Finally, we are investigating algorithms for the automatic generation of variational implicit surface models for the anatomic structures currently identified as part of the Visible Human Project (VHP) [9]. Four hundred thirty-five (435) hand-segmented structures comprise the current segmented thorax database, with each segment modeled using an uncompressed binary bitmask. For example, the bitmask for the heart is over 400 kilobytes, alone. We are seeking a means of automatically generating compact analytical descriptions of these models using the techniques described here.

## 6. Conclusions

Given a set of surface points  $C = \{\bar{c}_1, \bar{c}_2, \bar{c}_3, \dots, \bar{c}_n\}$ , the variational implicit technique interpolates the smoothest possible scalar function  $f(\bar{x})$  whose zero level set,  $\{\bar{x}: f(\bar{x}) = 0\}$ , passes through all points in  $C$ . These surface constraints or points can be generated from a variety of sources including segmentation systems. We show that variational implicit surfaces can be effectively used to model anatomic structures. Earlier limitations of computational and memory efficiency can be solved through a judicious selection of interpolant and improved numerics. Examples including noisy data from modalities generating curvilinear gridded data, such as endovascular ultrasound, demonstrate the utility of this technique.

## 7. Acknowledgements

This work was performed in large part at the National Library of Medicine under a visiting faculty program supporting both Dr. Morse and Dr. Subramanian. Dr. Rheingans was supported in part by NSF CAREER Grant #9996043. We would like to thank Greg Turk for his useful conversations and for making his code available to us, upon which our implementation is based.

## References

- [1] Bookstein, F. L. 1991. Morphometric tools for landmark data. Cambridge University Press
- [2] Duchon, J. 1977. Splines minimizing rotation-invariant semi-norms in Sobolev spaces, in Constructive theory of functions of several variables, Lecture Notes in Mathematics, edited by A. Dolb and B. Eckmann, Springer-Verlag, 1977, pp. 85–100.
- [3] Floater, M. S. and A. Iske. 1996. Multistep scattered data interpolation using compactly supported radial basis functions. J. Comp. Appl. Math. 73, pp 65-78.
- [4] Gibson, S. 1998. Constrained elastic surface nets: generating smooth surfaces from binary segmented data, in Proceedings of Medical Image Computing and Computer Assisted Interventions (MICCAI 1998)W. M. Wells, A. Colchester, and S. Delp, *eds.*, Lecture Notes in Computer Science 1496, Springer-Verlag, pp. 888-898.
- [5] Gibson, S. 1998. Using distance maps for accurate surface representation in sampled volumes, in Proceedings of the 1998 Symposium on Volume Visualization, ACM SIGGRAPH, pp. 23-30.
- [6] Kaufman, A. Volume visualization. IEEE Computer Society Press, Los Alamitos, CA, 1991.



- [7] Lorensen, W. and H. Cline. 1987. Marching Cubes: a high-resolution 3D surface construction algorithm. In Proc. SIGGRAPH 87, Computer Graphics, 21(4), pp. 163-169.
- [8] Morse, B., T. Yoo, P. Rheingans, D. Chen, and K.R. Subramanian. 2000. Complex Models Using Variational Implicit Surfaces. Submitted to Shape Modeling International 2001.
- [9] V. Spitzer, M. J. Ackerman, A. L. Scherzinger, and D. Whitlock. 1996. The Visible Human Male: A Technical Report. J. of the Am. Medical Informatics Assoc. 3(2) 118-130.
- [10] Szeliski, R. 1990. Fast surface interpolation using hierarchical basis functions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(6):513-528, June 1990.
- [11] Turk, G. and J. F. O'Brien. 1999. Variational implicit surfaces, Tech Report GIT-GVU-99-15, Georgia Institute of Technology, May 1999, 9 pages.
- [12] Turk, G. and J. F. O'Brien. 1999. Shape transformation using variational implicit functions. Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 99), pp. 335-342.
- [13] Wendland, H. 1995. Piecewise polynomial positive definite and compactly supported radial basis functions of minimal degree. AICM 4 (1995), pp. 389-396.
- [14] Whitaker, R. 2000. Reducing Aliasing Artifacts in Iso-Surfaces of Binary Volumes. in Volume Visualization and Graphics Symposium 2000, ACM SIGGRAPH, pp. 23-32.

# Active Contours Using a Constraint-Based Implicit Representation

Bryan S. Morse<sup>1</sup>, Weiming Liu<sup>1</sup>, Terry S. Yoo<sup>2</sup>, Kalpathi Subramanian<sup>3</sup>

<sup>1</sup>*Department of Computer Science, Brigham Young University*

<sup>2</sup>*Office of High Performance Computing and Communications, National Library of Medicine*

<sup>3</sup>*Department of Computer Science, The University of North Carolina at Charlotte*

## Abstract

We present a new constraint-based implicit active contour, which shares desirable properties of both parametric and implicit active contours. Like parametric approaches, their representation is compact and can be manipulated interactively. Like other implicit approaches, they can naturally adapt to non-simple topologies.

Unlike implicit approaches using level-set methods, representation of the contour does not require a dense mesh. Instead, it is based on specified on-curve and off-curve constraints, which are interpolated using radial basis functions. These constraints are evolved according to specified forces drawn from the relevant literature of both parametric and implicit approaches.

This new type of active contour is demonstrated through synthetic images, photographs, and medical images with both simple and non-simple topologies. For complex input, this approach produces results comparable to those of level set or parameterized finite-element active models, but with a compact analytic representation. As with other active contours they can also be used for tracking, especially for multiple objects that split or merge.

## 1. Introduction

Active contour models (also known as *deformable contours* or *snakes*) have been used prominently throughout computer vision since their introduction [9]. These models are iteratively updated according to various forces designed to seek out object/region boundaries while maintaining smoothness of the fitted contour, as shown in Figure 1. In this way, active contours provide a robust tool for image segmentation: the boundary-seeking portion of the model (*external energy*) provides the segmentation while the smoothness-preserving portion (*internal energy*) regularizes noisy data and handles missing or low-confidence sections of the contour. Interactively controlled forces may also be introduced to allow the user to guide the segmentation. This robustness has made active contours particularly popular for medical imaging applications, as surveyed

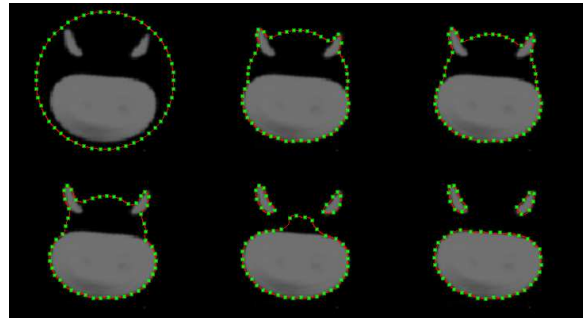


Figure 1: A constraint-based implicit active contour used to segment the multiple disjoint parts of a vertebrae cross-section. Although initialized as a single encompassing circle, the active contour changes its topology naturally to adapt to the disjoint parts. The green points denote the evolving constraints, and the red curves are the evolving contours defined by those constraints.<sup>1</sup>

in [14]. Active contour models have also proven useful for object tracking, both for medical imaging and other applications, because of their ability to update their position and shape as the segmented object moves.

For many problem domains it is necessary for an active contour to be able to adapt to non-simple topologies (as in Figure 1). This includes situations where a single object has holes and it is necessary for the active contour to wrap to both the interior and exterior contours of the object; or it might include situations where a single structure branches as it is tracked through 2-D slices of a volumetric image, causing two disjoint structures that need to be tracked in subsequent slices. Without this ability to split (or, going the other way, to merge), only one branch could be tracked.

Most implementations of active contours use *parametric models*, splines or other interpolants defined by a sequence (2-D) or mesh (3-D) of control points. Because of their reliance on a parametric representation, simple implementations of active contours cannot adapt to non-simple topologies. Multiple active contours can be used to segment

<sup>1</sup>This and other figures in this paper use color to convey the different components of the active contour. If these are not distinguishable in the printed copy, please refer to the PDF copy of this paper if available.

non-simple topologies, but the topology must be known and fixed. A topologically-adaptive variation of active contours known as *T-snakes* [13, 15] addresses this problem by selectively splitting or merging active contours periodically, then allowing them to continue to relax towards a solution. This approach is effective, but the topology changes only through periodic testing and reparameterization, not as a natural part of the representation. Because of the parametric nature of the representation, though, T-snakes are able to make use of existing techniques from the parametric snake literature, including user control.

Another approach to segmenting non-simple topologies is to use an *implicit* representation. Implicit contours or surfaces are defined as a level set (usually the zero set) of an *embedding function* whose domain is the space in which the contour or surface is represented (usually the image plane or volume). Because there is no explicit parameterization, implicit representations can have arbitrarily complex topologies while still using a topologically simple embedding function. Implicit active contours [3, 4, 10, 21, 30, 31] can thus segment and track topologically non-simple objects. Implicit active contour implementations typically represent the embedding function using a dense mesh of values, often corresponding to the image’s own pixel grid. These are then updated iteratively using *level-set methods* [20, 21, 22, 24, 25] so as to cause the zero set (the curve or surface) to move as desired. The PDEs (forces) driving the movement of the implicit curve or surface generally correspond to the traditional energy terms in parametric approaches. (See [35] for a comparison of the two approaches.) As the embedding function changes according to these PDEs, the topology of the implicitly represented active curve or surface can change naturally without being explicitly tested or changed. Unlike T-snakes, this topological adaptation occurs as a normal part of the active contour’s iteration. However, implicit active contours using dense meshes (even those using narrow-band [1, 12], fast marching methods [25], or sparse-field methods [32]) require storage and calculation for a large number of points.

We propose a new form of implicit active contour that uses a *sparse, compact* representation like parametric approaches but has the ability to adapt to complex topologies like other implicit approaches. This is based on a relatively new form of implicit contour representation that uses point-based *constraints* (analogous to control points) to define and control the curve or surface [2, 6, 7, 17, 19, 23, 26, 28, 29]. We call this new model a *constraint-based implicit active contour*. In some ways it shares similarities with the particle-based approach of [27] but evolves according to *surface* rather than *particle* particles. Like other implicit active contours, there is no finite-element representation, so it can easily adapt to non-simple or changing topologies.

Like parametric active contours, though, the representation is compact.

## 2. Constraint-Based Implicit Representations

Implicit curves or surfaces need not be represented by dense representations. One can use sparse primitives (usually “blobby” or medial structures), which provide a much more compact representation but don’t allow the same degree of control directly over the curve or surface. One can also use algebraic surfaces, but these quickly become too complicated for complex surfaces unless one subdivides the surface into patches.

Since the mid-90s, a number of techniques have emerged using scattered data interpolation techniques (most commonly *radial basis functions* or RBFs) to interpolate implicit curves, surfaces, or hypersurfaces from scattered points and some number of additional off-curve constraints [2, 6, 7, 17, 19, 23, 26, 28, 29]. Since the constraints are directly on the curve, these techniques give a much greater degree of control than the sum-of-primitives approach; and since they use only a scattered set of constraints, they are much more compact representations than dense meshes. These techniques have gone by various names in the literature, including *variational implicit surfaces* when constructed as a variational problem, *implicit surfaces that interpolate*, etc. We prefer the term *constraint-based implicit* curve or surface due to the reliance on scattered surface constraint points.

These constraint-based methods basically take the same approach: known points on the curve define points where the implicit curve’s embedding function should have a value of zero, known off-curve constraints define points where the embedding function has nonzero values, and these (point,value) targets are then interpolated using scattered data interpolation. Though they differ in various ways (the interpolation used, the means of defining the off-curve constraints, and the tolerance of fitting the points), they all share this key idea: rather than explicitly interpolating the curve or surface, *they interpolate the embedding function that implicitly defines it*.

An example of this is shown in Figure 2. Zero-valued constraints define the curve and nonzero-valued constraints are uniformly distributed around the perimeter of the image. The scattered (point,value) pairs constraining the implicit curve or surface are then interpolated using *radial basis functions* (RBFs) as follows.<sup>2</sup>

We begin with a set of constraints  $(c_i, h_i)$  such that  $h_i = 0$  for all  $c_i$  on the curve and  $h_i = 1$  for all  $c_i$  known to

<sup>2</sup>We follow most closely the general approach outlined in [29] and used in [17], [6], and related works. The description of the process here is intentionally brief, and we encourage the interested reader to consult these more detailed descriptions.

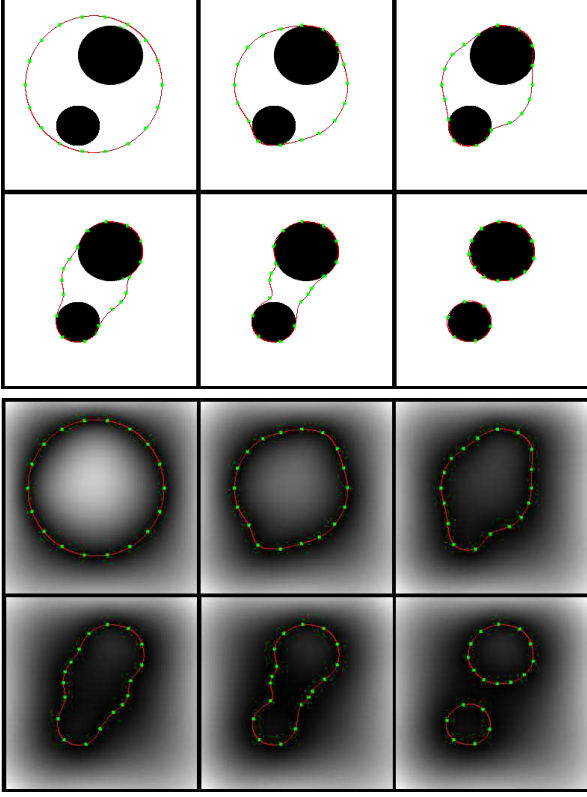


Figure 2: A constraint-based implicit active contour. As the constraints defining the contour evolve towards the object boundaries (top), the contour separates into two parts, occurring naturally as part of its underlying implicit representation (bottom). For visualization, we here show the absolute value of that embedding function, which may best be interpreted as an approximate distance field.

lie away from the curve. We then interpolate an embedding function  $f$  from these constraints such that  $\forall i : f(\mathbf{c}_i) = h_i$ .

We can obtain this interpolation using an RBF  $\phi(r)$  by defining the embedding function  $f$  as a weighted sum of these basis functions centered at each of the constraint positions, plus possibly an additional polynomial  $p$  (required for some basis functions):

$$\phi(\mathbf{x}) = \sum_{i=1}^n d_i \phi(\|\mathbf{x} - \mathbf{c}_i\|) + p(\mathbf{x}) \quad (1)$$

where  $\mathbf{c}_i$  is the position of the constraint and  $d_i$  is the weight of the radial basis function positioned at that point.

To solve for the set of weights  $d_i$  that satisfy the known constraints  $f(\mathbf{c}_i) = h_i$ , we substitute each constraint  $(\mathbf{c}_i, h_i)$  into Eq. 1:

$$\phi(\mathbf{c}_i) = \sum_{j=1}^n d_j \phi(\|\mathbf{c}_i - \mathbf{c}_j\|) + p(\mathbf{x}) = h_i \quad (2)$$

Eq. 2 thus defines a system of equations for solving for the weights in Eq. 1, which can now be used as an embedding function implicitly defining a smooth curve passing through the known constraints.

Constraint-based implicit curves or surfaces have already demonstrated themselves to be valuable for shape modeling [29], shape interpolation [28], surface reconstruction [2, 6], and medical imaging [36].

### 3. Constraint-Based Implicit Active Contours

We propose that constraint-based implicit curves or surfaces also provide an implicit representation suitable for implicit active contours. This representation is much more compact than previous dense-mesh or volumetric approaches. It can be stored using only the constraint points and the results of solving for the weights in Eq. 2, and the embedding function implicitly defining the curve or surface can be reconstituted analytically using Eq. 1.

#### 3.1. Basic Formulation

As with all active contour algorithms, we initialize the active contour based on an initial estimate of the object's shape and position. This could be based on an anatomical atlas, the results of segmenting a previous slice or frame, or simply a standard starting point such as a simple circle.

We place along an approximate initial curve a number of zero-valued constraints as described in Section 2. We also place along the image border a number of nonzero-valued constraints to define the exterior of the object. (These points could be placed arbitrarily distant from the center of the image, but we have chosen to include them in the image so that they may be better visualized.) We then solve the system of equations required for the RBF interpolation (Eq. 2) in order to build the embedding function  $\phi$  that implicitly defines our initial active contour.

We then adjust our curve constraints according to a number of energy functionals designed to move the contour towards the desired solution. (The nonzero constraints remain as a bounding box or circle around the object and are not updated.) A similar approach to evolving constraint-based implicit surfaces can also be found in [18].

For the basic implementation, we use an external image force  $F_{\text{image}}$ , an internal smoothing force  $F_{\text{internal}}$ , a balloon force  $F_{\text{balloon}}$ , and an internal constraint repulsion force  $F_{\text{repulse}}$ . Together, these forces drive the evolution of the constraints:

$$\begin{aligned} \frac{\partial}{\partial t} \mathbf{c}_i &= w_{\text{image}} F_{\text{image}}(\mathbf{c}_i) \\ &+ w_{\text{internal}} F_{\text{internal}}(\mathbf{c}_i) \\ &+ w_{\text{balloon}} F_{\text{balloon}}(\mathbf{c}_i) \\ &+ w_{\text{repulse}} F_{\text{repulse}}(\mathbf{c}_i) \end{aligned} \quad (3)$$

The weights of these forces ( $w_{\text{image}}$ ,  $w_{\text{internal}}$ ,  $w_{\text{balloon}}$ , and  $w_{\text{repulse}}$ , respectively) may be adjusted to control the relative importance of each component.

Another approach, rather than using separate external image and balloon forces to drive motion, borrows a technique from the level-set implicit snake literature [3]. This approach uses a balloon force to externally drive the motion, the internal force to induce smoothness, and a boundary potential “stopping function”:

$$\begin{aligned} \frac{\partial}{\partial t} \mathbf{c}_i &= g(\mathbf{c}_i) [w_{\text{internal}} F_{\text{internal}}(\mathbf{c}_i) + w_{\text{balloon}} F_{\text{balloon}}(\mathbf{c}_i)] \\ &+ w_{\text{repulse}} F_{\text{repulse}}(\mathbf{c}_i) \end{aligned} \quad (4)$$

where the stopping function  $g(\mathbf{c}_i)$  is a function of the image boundary potential ranging in value from 1 (no external force, let the balloon and internal forces drive the motion) to 0 (on a boundary, stop).

These individual forces are defined as follows, specifically as in Eq. 5 through Eq. 9.

### Image Energy Force

As in parametric active contour models, we define an image boundary potential function  $P(\mathbf{x})$  that is low for points with high boundary-like properties. We then define one of the terms driving the motion of the constraints as the negative gradient of this potential:

$$F_{\text{image}} = -\nabla P$$

Moving the constraints only along the normal to the implicit curve (as effectively done by level-set based implicit active contour algorithms) gives a more effective constraint motion. Denoting the curve’s normal as

$$N = \frac{\nabla \phi}{\|\nabla \phi\|}$$

this becomes

$$F_{\text{image}} = -(\nabla P \cdot N) N \quad (5)$$

We can use any of the external energy functionals in the existing literature for parametric snakes and have implemented such variants as gradient vector flow [34].

### Internal Energy Force

For the internal energy term, we borrow not from parametric active contours but from their implicit counterparts. Implicit active contours use differential geometry and the derivatives of the embedding function to calculate the curvature of the level set representing the curve. Using level-set methods, the embedding function is then adjusted so as to reduce the curvature of the implicitly represented curve (*mean curvature flow*).

We also measure the curvature of the active contour by using differential geometry to calculate the curvature  $\kappa = \text{div} \frac{\nabla \phi}{\|\nabla \phi\|}$  of the level set passing through each constraint along the curve. We then explicitly move each constraint in the direction of the curve’s local normal at a rate proportional to the negative of the local curvature:

$$F_{\text{internal}} = -\kappa N \quad (6)$$

### Balloon Force

Balloon forces can be used to make the active contour work like a balloon: expanding when inside the shape boundaries and shrinking when outside the shape boundaries in the normal direction of the curve [5, 3, 16].

The motion due to the balloon force  $F_{\text{balloon}}(\mathbf{c}_i)$  at constraint  $i$  can be expressed as

$$F_{\text{balloon}}(\mathbf{c}_i) = F(I(\mathbf{c}_i)) N(\mathbf{c}_i) \quad (7)$$

For images whose shape regions have different intensity from the background and can be segmented using a simple threshold  $T$ ,  $F(I(\mathbf{c}))$  is simply  $\pm 1$  depending on whether the image intensity  $I(\mathbf{c})$  is above or below threshold.

For more complex distributions of intensities in the image, we can use information about the region intensity statistics [8]. Assuming that the shape regions have intensity mean  $\mu$  and standard deviation  $\sigma$ , and  $k$  is a user-adjustable constant,  $F(I(\mathbf{c}))$  can be designed as

$$F(I(\mathbf{c})) = \begin{cases} +1 & \text{if } |I(\mathbf{c}) - \mu| \leq k\sigma, \\ -1 & \text{otherwise} \end{cases} \quad (8)$$

The constraint-based implicit representation makes it easy to determine the statistics of the enclosed region(s), because *the embedding function acts as a characteristic object-membership function for all pixels in the image*.

### Constraint Repulsion Force

To encourage uniform distribution of the constraints along a contour, we add an additional motion term that acts to push constraint points apart and leads to roughly uniform spacing [33]. We model this energy term after electrostatic potential between charged particles. If we think of on-curve constraints as unit positive charged particles and ignore momentum, the combined repulsive force on the  $i$ th constraint due to other constraints is

$$F_{\text{repulse}}(\mathbf{c}_i) = \sum_{j \neq i} \frac{\mathbf{c}_i - \mathbf{c}_j}{\|\mathbf{c}_i - \mathbf{c}_j\|^3}$$

Since the repulsive force becomes unstable as the distance between the points becomes very small, we can also approximate this using a Gaussian-based function of the distance as in [33].

To avoid this repulsive force acting as a secondary ballooning force, we constrain the effect of the repulsion to be only in the tangent to the curve ( $N_\perp$ ). To avoid interaction between disjoint curves once the topology changes, we also weight the repulsive force between two points by the similarity between the normals at those points:

$$F_{\text{repulse}}(\mathbf{c}_i) = \sum_{j \neq i} \left[ w_{ij} \frac{(\mathbf{c}_i - \mathbf{c}_j)}{\|\mathbf{c}_i - \mathbf{c}_j\|^3} \cdot N_\perp(\mathbf{c}_i) \right] N_\perp(\mathbf{c}_i) \quad (9)$$

where  $w_{ij} = \frac{1}{2} [1 + (N(\mathbf{c}_i) \cdot N(\mathbf{c}_j))]$ .

### 3.2 Implementation

We implement the basic constraint-based implicit active contour algorithm as follows:

1. Preprocess the original image by blurring with a Gaussian to reduce noise, make edges cleaner, and increase the capture range for the active contour.
2. Select a set of initial constraints around the shape of interest. This may be done by having the user supply an initial estimate of the contour; or they may be drawn from a prior model of the shape or from a previous slice or frame.
3. Construct an embedding function from these constraints using thin-plate spline radial basis functions and the methods described in Section 2. Generally it is better to select constraints near the desired boundaries, which then require fewer iterations to find the final boundaries. However, our model also allows the user to select constraints farther away from the boundaries.
4. Evolve the constraints according to Eq. 3 or Eq. 4 for 5–10 time steps. During this process, we use the same embedding function because it changes little.
5. Reconstruct the embedding function from the changed constraints after each set of 5–10 time steps using an incremental solver (an iterative solver that uses the previous solution as a starting point).
6. Stop evolving when the active contour reaches object boundaries and converges.

At no time during the algorithm do we need to extract the contour from its implicit representation or to otherwise use any form of finite-element, spline, or other explicit representation. (In our implementation we do so only to provide visualization of the intermediate steps of the evolution.)

A direct solver can be used for Step 5, but using an incremental solver takes advantage of the RBF weights calculated for the previous embedding function, usually converging to the new solution within only a few iterations.

We use a base time step of  $\Delta t_0 = \frac{1}{\max(w_{\text{internal}}, w_{\text{balloon}})}$ . We then conservatively select a time step so as to limit the motion of a single constraint to be no more than one pixel:  $\Delta t = \frac{\Delta t_0}{\max_i \|\mathbf{c}_i^+ - \mathbf{c}_i^t\|}$ .

### 3.3 Enhancements

#### Inserting and Deleting Constraints

Many snake implementations insert additional constraints as the curve expands. Although we have no explicit parameterization, we can likewise insert or delete constraints by recognizing when a constraint becomes too far from or too close to nearby constraints [33]. This pairwise constraint-to-constraint distance calculation requires no additional computation because it is already part of the RBF calculations. If the minimum distance from one constraint to all other constraints exceeds a predetermined threshold, we split that constraint into two new constraints placed at a small offset in the curve's tangent direction from the original. If the minimum distance becomes too small, we merge those constraints. This is useful in avoiding instabilities in the repulsive forces when collapsing to a small object.

#### User Interaction

As with the original snake implementation [9], we can also introduce “springs”: user-defined forces to pull a specific constraint ( $\mathbf{c}_i$ ) towards a goal ( $\mathbf{a}$ ):

$$F_{\text{spring}} = (\mathbf{a} - \mathbf{c}_i)^p$$

for some exponent  $p$ , or to push them all away from that point (“volcanos”):

$$F_{\text{volcano}} = \frac{1}{\|\mathbf{a} - \mathbf{c}_i\|^3} [(\mathbf{c}_i - \mathbf{a}) \cdot N(\mathbf{c}_i)] N(\mathbf{c}_i)$$

See Figure 7 for an example of the application of user interaction to snake evolution.

#### Automatic Constraint Addition

In some cases, the user may wish to indicate that the snake is missing a significant part of the object (or because of the topological flexibility, a disjoint part). This can be accomplished by adding new constraints, which can be placed automatically by finding points in the image where both the object boundary likelihood and the distance from the current snake is high. Using the negative of the boundary potential  $-P(\mathbf{x})$  and recognizing that the embedding function can serve as a pseudo-distance field, we can define this as the point that maximizes  $-P(\mathbf{x}) |\phi(\mathbf{x})|$ .

In our implementation, we found that a pseudo-distance field is better created by non-zero constraints placed at a fixed offset from the zero-valued constraints [2]. Since we do not do this during normal snake evolution, we do so only when asked to automatically add new constraints.



## 4. Results

Figures 3–4 show results for simple synthetic images in order to demonstrate how constraint-based implicit snakes work. Figure 3 shows a single contour adapting to multiple disjoint objects, and Figure 4 shows several initial contours merging to segment the separate interior and exterior boundaries of a hollow shape.

Figure 1 and Figures 5–8 show the use of constraint-based implicit active contours to segment various types of medical images. Figure 1 shows how the contour can change topology to adapt to multiple disjoint pieces of an object. Figures 5–7 show how even in situations with simple topology, constraint-based implicit active contours perform in ways comparable to both parametric or level-set based methods. Figure 7 also demonstrates user interaction to direct the contour away from an interfering nearby boundary. Figure 8 shows a complex branching (though topologically simple) shape, which can be segmented using a single constraint-based implicit active contour through the use of automatic point insertion as the contour grows.

Finally, Figures 9–10 show how constraint-based implicit snakes can be used to track objects in video sequences. In each, the result for one frame is used as the initial estimate for the following frame. In particular, Figure 10 demonstrates tracking multiple objects as they merge and later separate.

## 5. Conclusion

We have presented a new approach to topology-adaptive active contours using a constraint-based implicit representation. Like parametric active contours, the representation uses only a sparse number of points on the contour. Like other implicit active contours, topological changes happen naturally as part of their implicit representation. These new constraint-based implicit active contours thus combine the best features of both implicit approaches (naturally topology-adaptive) and parametric approaches (compact representations, user interaction).

Examples have been shown for simple synthetic images, photographs, medical images, and video sequences. These examples show that in cases of simple topology, constraint-based implicit active contours perform in ways comparable to either parametric or level-set based approaches. In cases with more complex topologies, constraint-based implicit active contours adapt naturally to the topology in ways comparable to level-set based methods or T-snakes. However, the representation requires neither the dense mesh required for level-set methods nor the ACID node structure required for T-snakes. The representation is compact and can be analytically defined by simply listing the (unordered) constraints that evolve to localize the object.

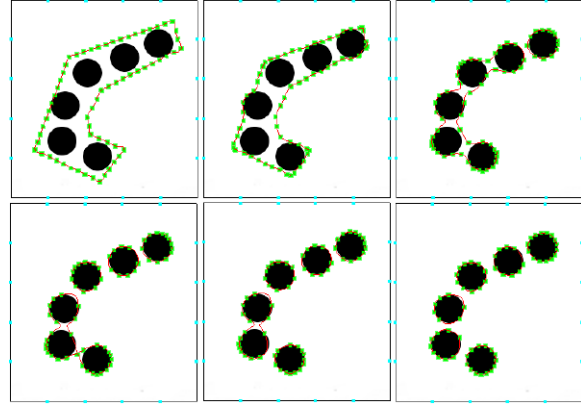


Figure 3: Synthetic image with one initial contour and six disjoint targets. As the contour evolves, it breaks naturally into multiple disjoint curves.

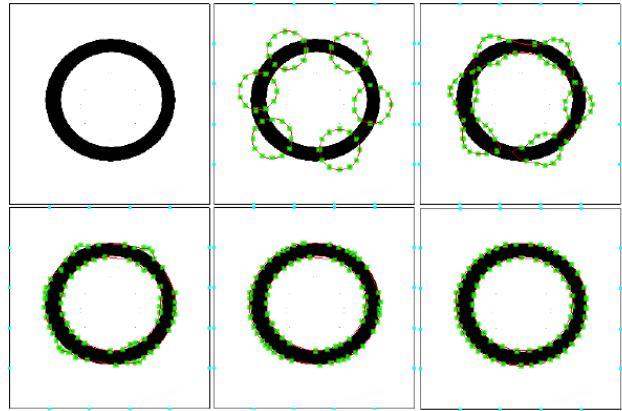


Figure 4: Six initial contours merging to form two contours, one each for the interior and exterior boundaries.

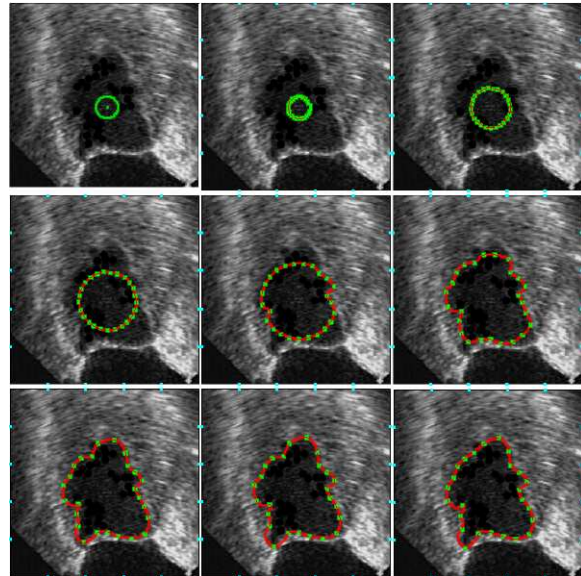


Figure 5: Segmentation of the left-ventricular chamber of the heart (LV) in an ultrasound image.

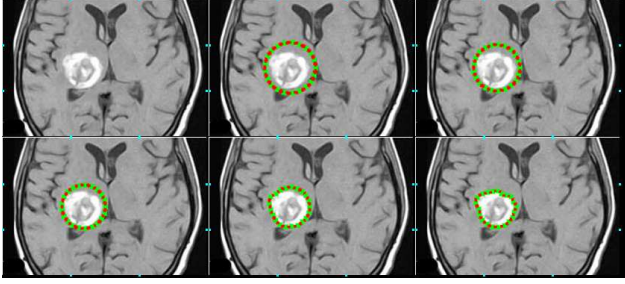


Figure 6: Segmentation of a tumor in a slice of an MRI using a constraint-based implicit active contour.

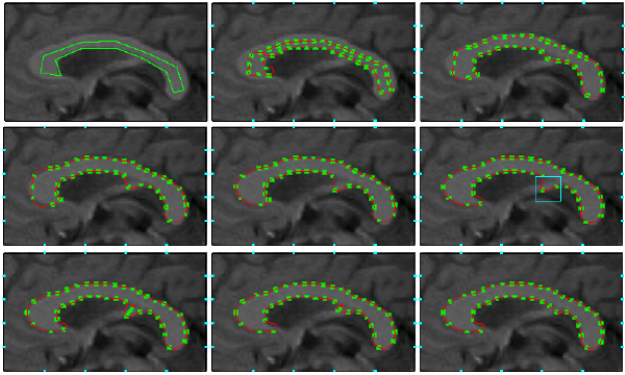


Figure 7: Segmentation of the corpus callosum. A user-defined “spring” (indicated with a red dot for the anchor) is placed interactively to correctively pull the contour away from a nearby boundary.

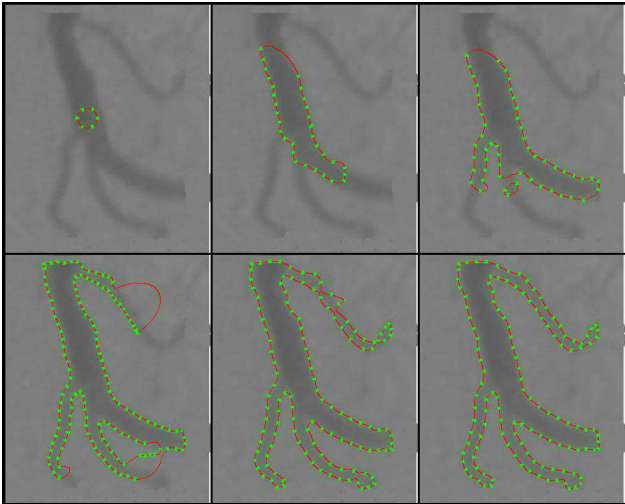


Figure 8: Segmentation of a blood vessel with complex branching structure. Although initialized with a small circle in the interior of the vessel, the active contour expands to segment the entire structure. As the contour expands, additional constraints are inserted automatically even though there is no parameterization or even ordering of the points.

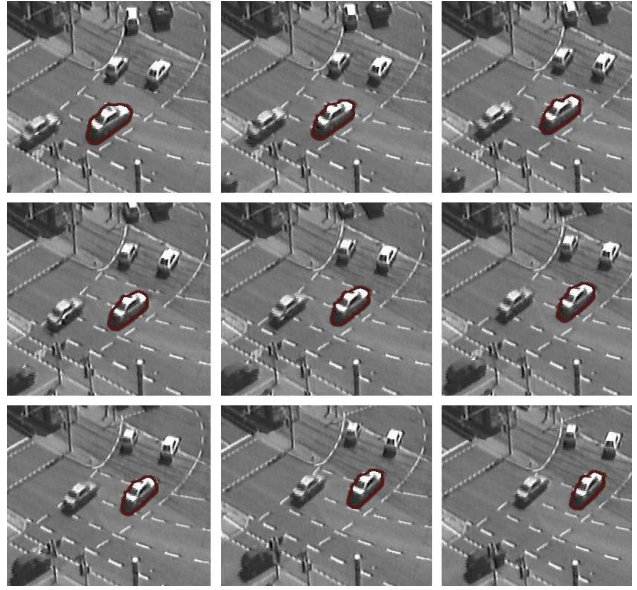


Figure 9: Using constraint-based implicit active contours to track a car in a traffic sequence. As is commonly done, the active contour for each frame was initialized using the results of the previous frame.

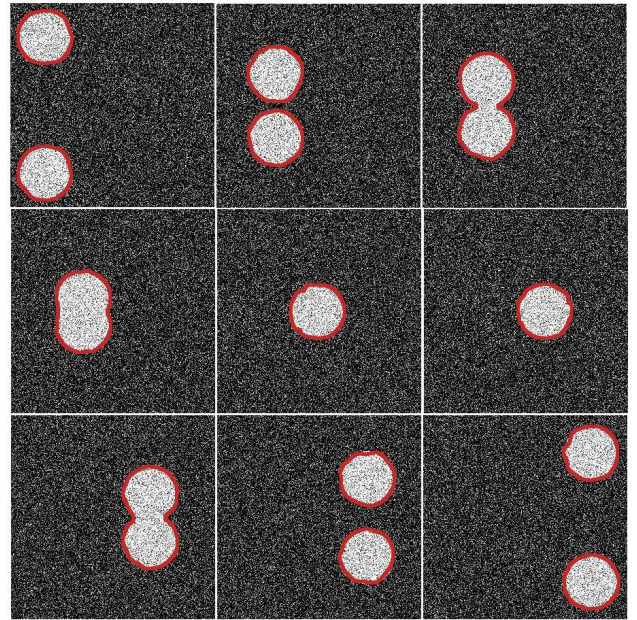


Figure 10: Constraint-based implicit active contour tracking multiple objects through a synthetic motion sequence (top-to-bottom, left-to-right). As the objects approach each other and combine, their respective active contours merge implicitly. As the objects later separate again, their respective active contours also implicitly separate.



## Acknowledgments

We would like to thank Lauralea Otis, David Chen, and Tom Sederberg for their help with this work. We would also like to thank Greg Turk, James O'Brien, Quynh Dinh, Ross Whitaker, and John Hart for their useful discussions regarding implicit surface modeling.

## References

- [1] D. Adalsteinsson and J. Sethian. A fast level set method for propagating interfaces. *J. Computational Physics*, 118:269–277, 1995.
- [2] J. C. Carr, T. J. Mitchell, R. K. Beatson, J. B. Cherrie, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis. In *SIGGRAPH 2001 Proceedings*, Annual Conference Series. ACM SIGGRAPH, ACM Press, August 2001.
- [3] V. Caselles, F. Catté, B. Coll, and F. Dibos. A geometric model for active contours in image processing. *Numerische Mathematik*, 66(1), 1993.
- [4] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *In Proc. Fifth International Conf. on Computer Vision (ICCV'95)*, pages 694–699, Los Alamitos, CA, June 1995. IEEE Computer Society Press.
- [5] L. D. Cohen. On active contour models and balloons. *CVGIP: Image Understanding*, 56(2):242–263, 1991.
- [6] H. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, October 2002.
- [7] H. Q. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces using anisotropic basis functions. In *Proceedings Eighth International Conference on Computer Vision (ICCV 2001)*, 2001.
- [8] J. Ivins and J. Porrill. Statistical snakes: Active region models. In *Proceedings Fifth British Machine Vision Conference (BMVC'04)*, pages 377–386, 1994.
- [9] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.
- [10] S. Kichenassamy, A. Kumar, P. Older, A. Tannenbaum, and A. Yezzi. Gradient flows and geometric active contour models. In *In Proc. Fifth International Conf. on Computer Vision (ICCV'95)*, pages 810–815, Los Alamitos, CA, June 1995. IEEE Computer Society Press.
- [11] W. Liu. Constraint-based implicit snakes using thin-plate spline radial basis functions. Master's thesis, Brigham Young University, April 2004.
- [12] R. Malladi, J. Sethian, and B. C. Vemuri. Shape modeling with front propagation: a level-set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, Feb. 1995.
- [13] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *Proceedings Fifth International Conference on Computer Vision*, pages 840–845. IEEE Computer Society Press, June 1995.
- [14] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Imaging Analysis*, 1(2), 1996.
- [15] T. McInerney and D. Terzopoulos. Topologically adaptive deformable surfaces for medical image volume segmentation. *IEEE Trans. Medical Imaging*, 20:100–111, 1996.
- [16] T. McInerney and D. Terzopoulos. T-snakes: Topology adaptive snakes. *Medical Image Analysis*, 4:73–91, 2000.
- [17] B. S. Morse, T. S. Yoo, D. T. Chen, P. Rheingans, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings Shape Modeling International*, 2001.
- [18] M. Mullan, R. Whitaker, and J. Hart. Procedural level sets. Presented at the NSF/DARPA CARGO meeting, May 2004.
- [19] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicit. In *Proceedings 2003 SIGGRAPH*, Annual Conference Series. ACM SIGGRAPH, ACM Press, 2003.
- [20] S. Osher and R. Fedkiw. *Level-Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag New York, Inc., 2003.
- [21] S. Osher and N. Paragios. *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer-Verlag New York, Inc., 2003.
- [22] S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation. *J. Comput. Phys.*, 79:12–49, 1988.
- [23] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.
- [24] J. A. Sethian. *Level Set Methods*. Cambridge University Press, 1996.
- [25] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.
- [26] C. Shen, J. F. O'Brien, and J. R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings 2004 SIGGRAPH*, Annual Conference Series. ACM SIGGRAPH, ACM Press, 2004.
- [27] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *Proceedings Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society Press, 1993.
- [28] G. Turk and J. F. O'Brien. Shape transformation using variational implicit surfaces. In *SIGGRAPH '99 Proceedings*, Annual Conference Series. ACM SIGGRAPH, ACM Press, 1999.
- [29] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, October 2002.
- [30] J. Weickert and G. Kühne. Fast methods for implicit active contour models. In S. Osher and N. Paragios, editors, *Geometric Level Set Methods in Imaging, Vision, and Graphics*, pages 43–77. Springer-Verlag New York, Inc., NY: New York, 2003.
- [31] R. Whitaker. Volumetric deformable models: active blobs. In R. Robb, editor, *Visualization in Biomedical Computing*, pages 122–134, November 1994.
- [32] R. T. Whitaker. A level-set approach to 3D reconstruction from range data. *International Journal of Comp. Vision*, 10(3):203–231, 1998.
- [33] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics*, 28(Annual Conference Series):269–277, 1994.
- [34] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. on Image processing*, 7(3):359–369, 1998.
- [35] C. Xu, A. Yezzi, and J. Prince. A summary of geometric level-set analogues for a general class of parametric active contour and surface models. In *Proc. of 1st IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 104–111, July 2001.
- [36] T. S. Yoo, B. S. Morse, K. R. Subramanian, P. Rheingans, and M. J. Ackerman. Anatomic modeling from unstructured samples using variational implicit surfaces. In *Proceedings of Medicine Meets Virtual Reality (MMVR 2001)*, Jan. 2001.

# Using Particles to Sample and Control Implicit Surfaces

Andrew P. Witkin

Paul S. Heckbert

Department of Computer Science  
Carnegie Mellon University

## Abstract

We present a new particle-based approach to sampling and controlling implicit surfaces. A simple constraint locks a set of particles onto a surface while the particles and the surface move. We use the constraint to make surfaces follow particles, and to make particles follow surfaces. We implement *control points* for direct manipulation by specifying particle motions, then solving for surface motion that maintains the constraint. For sampling and rendering, we run the constraint in the other direction, creating *floaters* particles that roam freely over the surface. Local repulsion is used to make floaters spread evenly across the surface. By varying the radius of repulsion adaptively, and fissioning or killing particles based on the local density, we can achieve good sampling distributions very rapidly, and maintain them even in the face of rapid and extreme deformations and changes in surface topology.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling: *Curve, surface, solid, and object representations, Physically based modeling*; I.3.6 [Computer Graphics]: Methodologies and Techniques: *Interaction techniques*; G.1.6 [Numerical Analysis]: Optimization: *Constrained Optimization*.

**General Terms:** algorithms, design.

**Additional Key Words and Phrases:** physically based modeling, constrained optimization, adaptive sampling, interaction.

## 1 Introduction

Implicit surfaces have proven to be useful for modeling, animation, and visualization. One appeal of implicit models is that new surfaces can be created by adding or otherwise combining the functions that define them, producing a variety of subtle and interesting shape effects. Another is their role in the visualization of volume data. In addition, the implicit representation lends itself to such calculations as ray/surface intersection and inside/outside test. However, implicit surfaces suffer from two serious drawbacks: first, although well suited to ray tracing, they are not easily rendered at interactive speeds, reflecting the underlying problem that it is difficult to *sample* them systematically. This is particularly a problem if we wish to render time-varying surfaces in real time, which is vital for interactive sculpting. Second, the shapes of implicit surfaces have proven to be more difficult to specify and control than those of their parametric counterparts.

Mail to the authors should be addressed to the Department of Computer Science, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh PA 15213, USA. Email should be addressed to Andrew Witkin as [aw@cs.cmu.edu](mailto:aw@cs.cmu.edu), and to Paul Heckbert as [ph@cs.cmu.edu](mailto:ph@cs.cmu.edu).

©1994 ACM. Reprinted from *Computer Graphics*, Proc. SIGGRAPH '94.

In this paper, we present a new particle-based approach to sampling and shape control of implicit surfaces that addresses these problems. At the heart of our approach is a simple constraint that locks a collection of particles onto an implicit surface while both the particles and the surface move. We can use the constraint to make the surface follow the particles, or to make the particles follow the surface. Our formulation is differential: we specify and solve for velocities rather than positions, and the behavior of the system is governed by differential equations that integrate these velocities over time.

We control surface shape by moving particles interactively, solving for surface motion that keeps the particles on the surface. This technique lets us pin down the surface at some points while interactively dragging others. These particles act as *control points* for direct manipulation of the surface.

For sampling and rendering, we run the constraint in the other direction, creating particles that may roam freely over the surface, but are compelled to follow it as it moves. We call these particles *floaters*. Our starting point is the idea that uniform sampling density can be achieved by making the particles repel each other. This approach was used by Turk [29] to resample polygon meshes, and by Figueiredo *et al.* [12] to sample implicit surfaces.

Simple repulsion can work quite well for stationary surfaces, but only if a reasonably good initial sampling is available. If large-scale non-uniformities exist, convergence can be extremely slow for even moderate sampling densities. We eliminate the need for a good starting point, and dramatically accelerate convergence, by employing an iterative “fissioning” approach, in which we start with a small number of particles and a large radius of repulsion, allow them to equilibrate, then split each particle in two, reducing the radius of repulsion. We repeat the process until the desired sampling density is reached. Each level thus inherits a distribution that is already uniform at large scale, requiring just a few iterations to iron out the local irregularities.

Global fissioning still fails to handle surfaces that move and deform, since irregularities can arise after the density becomes high. We introduce a local adaptive repulsion algorithm, in which the repulsion radius and particle birth and death are regulated based on local density. This method is fast enough to maintain good sampling even in the face of rapid and extreme surface motion and deformation.

The remainder of the paper is organized as follows: we begin by discussing previous related work. Then we introduce the basic constraint mechanism that attaches particles to surfaces. Next we describe the use of particles for surface shape control. We then explain our adaptive repulsion sampling algorithm. After describing the implementation and results, we conclude with a discussion of future work.

## 2 Previous Work

Related work can be divided into two categories: sampling methods and control methods.

### 2.1 Sampling Methods

Related research on surface sampling includes both particle-based sampling techniques and polygonization techniques for implicit surfaces.

Turk used repelling particles on surfaces to uniformly resample a static surface [28] and to simplify a polygonization by reducing the number of polygons [29]. Hoppe *et al.* also explored mesh simplification, framing it as an optimization problem with penalties for geometric error, number of samples, and edge length [16]. Their method did not restrict the points to a surface, however, as Turk's and ours do.

Szeliski and Tonnesen used oriented particles to model surfaces [27]. Their technique allowed the user to move the particles interactively, employing short-range repulsion and long-range attraction to keep the particles from clumping or flying apart. The system generated a surface by connecting neighboring particles appropriately, but it did not manipulate a high level representation for a surface, such as a parametric patch or an implicit function, as ours does. The output of their system was a triangulation. Their system bears a superficial resemblance to ours because we both use disks to visualize the surface, but in other respects our techniques are quite different.

An implicit surface, also called an iso-surface, is the set of points  $\mathbf{x}$  that satisfy  $F(\mathbf{x}) = 0$ . Implicit surfaces are typically defined by starting with simple building block functions and by creating new implicit functions using the sum, min, or max of simpler functions. When the building blocks are polynomials in  $x$ ,  $y$ , and  $z$ , the resulting surfaces are called *algebraic surfaces*, and when the building blocks are spherical Gaussian functions, the surfaces are called *blobbies* [8], "soft objects", or "metaballs". The use of sums of implicit functions allows blend surfaces to be created [24], and the use of min and max yields the union and intersection of solid objects.

Rendering an implicit surface is often difficult. If a ray tracer is used, intersecting a ray with an implicit surface reduces to one-dimensional root-finding, but this can be very slow for a complex implicit function [8]. To exploit the speed of graphics hardware, we would prefer to render using a z-buffer algorithm. This requires converting the implicit surface into polygons or other parametric surfaces.

Most existing methods for polygonizing implicit surfaces subdivide space into a uniform grid of cubical or tetrahedral voxels, sample the function at the vertices of the voxels, and then, for each voxel whose vertices are not all in or all out, generate polygon(s) approximating the surface [33,18,21,10]. This approach is often called the *marching cubes* algorithm. Improvements on this algorithm use adaptive subdivision based on curvature [9]. Unfortunately, all of these algorithms will miss small surface features if the initial sampling grid is too coarse, except Snyder's, which uses interval arithmetic to guarantee that the topology of the polygonization matches the topology of the real surface [26].

These polygonization algorithms were designed for static surfaces; to polygonize a changing surface with them would require beginning from scratch each time. The algorithm of Jevans *et al.* is an exception. It re-polygonizes only those voxels that change [17].

Physically-based approaches to the polygonization of implicit surfaces were pioneered by Figueiredo *et al.* [12]. One of the two methods they describe starts with particles randomly scattered in 3-D space, subjects them to forces that pull them to the surface (an idea proposed in [11]), and uses repulsion between particles to distribute them uniformly over the surface. Their technique uses

penalty methods, however, which lead to stiff differential equations whose solution is generally either slow to repel into a nice pattern, or inaccurate at staying on the surface. Once the particles have reached equilibrium, a polygonization is found using Delaunay triangulation. Their work resembles ours most closely, but our simulation method differs from theirs, and our technique supports interactive control of surfaces and incremental sampling of changing surfaces, while theirs does not.

### 2.2 Control Methods

One of the principal disadvantages of implicit modeling relative to parametric modeling is the difficulty of controlling the shape of an implicit surface [11]. The effect of the parameters of an implicit surface is often non-intuitive.

With algebraic surfaces, for instance, it is hard to predict the surface shape given its coefficients. Modeling is further complicated by the global nature of an algebraic surface's polynomial basis functions, which prevent local shape control. For these reasons and others, piecewise algebraic surfaces have recently become popular [25]. Piecewise algebraic surfaces are typically defined by a weighted sum of Bernstein polynomials over a lattice of tetrahedra. Least squares methods for fitting surfaces to a set of points are available both for standard algebraic surfaces [22] and for piecewise algebraic surfaces [1]. Pratt's algorithm can fit a surface with  $m$  parameters to  $n$  points ( $n > m$ ) in time  $O((n+m)m^2)$ . These methods are limited to algebraic surfaces, however.

Bloppy models employ local basis functions, so they are often more intuitive to work with than algebraic surfaces [8]. In an interactive blobby modeling system, a user might use dials or sliders to adjust the position and radius of each blobby center [7], but arriving at a desired surface is a matter of guesswork, and the real time display is typically just a wireframe, with a higher quality rendering requiring off-line ray tracing or polygonization. Some recent work has fit blobby models to a set of surface points, but the method is quite slow, one example requiring days of computer time to fit 2900 control points using 1200 parameters [20]. Direct manipulation of a blobby surface at interactive speeds has remained an open problem.

The differential methods we use to constrain the motion of particles and surfaces are rooted in classical mechanics (see, e.g. [15] for a discussion of mechanical constraints and constraint forces) and are closely related to constraint methods used in physically based modeling for computer graphics [5,2,3,32,31,4]. Allied methods have also been used for interactive geometric modeling [30,14].

## 3 The Particle/Surface Constraint

In this section we derive the basic machinery that allows us to attach moving particles to moving surfaces. First we derive a basic constraint on particle and surface velocities that establishes, then maintains contact as the system evolves over time. We then pose two related problems: solve for particle velocities given time derivatives of the surface parameters, and solve for surface derivatives given particle velocities. Since the problem will generally be underconstrained, we express it as a constrained optimization.

**Notation:** We use boldface to denote vectors, and italics for scalars. Subscripts denote partial differentiation. Superscript  $i$  or  $j$  denote the  $i$ th or  $j$ th member of a collection of objects. E.g.  $\mathbf{p}^i$  is the  $i$ th in a collection of vectors, and  $F_{\mathbf{x}}$  is the derivative of scalar  $F$  with respect to vector  $\mathbf{x}$ , hence a vector. Superscripts other than  $i$  or  $j$  have their usual meaning as exponents, e.g.  $|\mathbf{x} - \mathbf{c}|^2$  or  $e^{-x^2}$ . A dot, as in  $\dot{\mathbf{q}}$ , denotes a derivative with respect to time.

### 3.1 The Basic Constraint

We represent the moving implicit surface by  $F(\mathbf{x}, \mathbf{q}(t)) = 0$ , where  $\mathbf{x}$  is position in space, and  $\mathbf{q}(t)$  is a vector of  $m$  time-varying shape parameters. For example, an implicit sphere could be defined by  $F = |\mathbf{x} - \mathbf{c}|^2 - r^2$ , with center  $\mathbf{c}$  and radius  $r$ . The parameter vector  $\mathbf{q}$  would then be the 4-vector  $[c_x, c_y, c_z, r]$ .

The condition that a collection of  $n$  moving particles lie on the surface is

$$F(\mathbf{p}^i(t), \mathbf{q}(t)) = 0, \quad 1 \leq i \leq n, \quad (1)$$

where  $\mathbf{p}^i(t)$  is the trajectory of the  $i$ th particle. In order for this condition to be met from some initial time  $t_0$  onward, it suffices that equation 1 is satisfied at  $t_0$ , and that the time derivative  $\dot{F} = 0$  thereafter. Since we want to manipulate velocities rather than positions, we obtain an expression for  $\dot{F}$  using the chain rule:

$$\dot{F}^i = F_{\mathbf{x}}^i \cdot \dot{\mathbf{p}}^i + F_{\mathbf{q}}^i \cdot \dot{\mathbf{q}}, \quad (2)$$

where  $\dot{F}^i$ ,  $F_{\mathbf{x}}^i$ , and  $F_{\mathbf{q}}^i$  denote  $\dot{F}$ ,  $F_{\mathbf{x}}$ , and  $F_{\mathbf{q}}$  evaluated at  $\mathbf{p}^i$ . By setting  $\dot{F}^i$  to zero in equation 2, we obtain  $n$  linear constraints on the  $\dot{\mathbf{p}}^i$ s and on  $\dot{\mathbf{q}}$ . In principle, if we began with a valid state and ensured that these conditions were met at every instant thereafter, we would be guaranteed that the particles remained on the surface. In practice, we might not have valid initial conditions, and numerical integration errors would cause drift over time. We cure these problems using a feedback term [6], setting  $\dot{F}^i = -\phi F^i$ , where  $\phi$  is a feedback constant. This yields the set of  $n$  linear constraint equations

$$C^i(\mathbf{p}^i, \dot{\mathbf{p}}^i, \mathbf{q}, \dot{\mathbf{q}}) = F_{\mathbf{x}}^i \cdot \dot{\mathbf{p}}^i + F_{\mathbf{q}}^i \cdot \dot{\mathbf{q}} + \phi F^i = 0 \quad (3)$$

### 3.2 Constrained Optimization

We employ these constraints in two ways: first, in order to use particles to move the surface, we solve for  $\dot{\mathbf{q}}$  given the  $\dot{\mathbf{p}}^i$ 's. Second, to use mutually repelling particles to sample the surface, we solve for the  $\dot{\mathbf{p}}^i$ 's given  $\dot{\mathbf{q}}$ . In either case, we generally wish to solve underconstrained systems. To do so we minimize a quadratic function of  $\dot{\mathbf{p}}^i$  and  $\dot{\mathbf{q}}$ , subject to the constraints. The objective function we use here is

$$G = \frac{1}{2} \sum_{i=1}^n |\dot{\mathbf{p}}^i - \mathbf{P}^i|^2 + \frac{1}{2} |\dot{\mathbf{q}} - \mathbf{Q}|^2,$$

where  $\mathbf{P}^i$  and  $\mathbf{Q}$  are known *desired* values for  $\dot{\mathbf{p}}^i$  and  $\dot{\mathbf{q}}$  respectively.<sup>1</sup> These desired values can be used in a variety of ways. Setting  $\mathbf{P}^i$  to zero minimizes particle velocities. Setting  $\mathbf{Q}$  to zero minimizes the surface's parametric time derivative.

In unconstrained optimization we require that the gradient of the objective function vanish. At a *constrained* minimum, we require instead that the gradient of the objective function be a linear combination of the gradients of the constraint functions [13]. This condition ensures that no further local improvement can be made without violating the constraints. In the case of a point constrained to a surface, this condition is easily visualized: the gradient of the objective function must lie normal to the surface, so that its orthogonal projection onto the tangent plane vanishes. Though harder to visualize, the idea is the same in higher dimensions.

<sup>1</sup>Although we do not give the derivation here, a straightforward and useful generalization is to allow error to be measured using an arbitrary symmetric positive-definite metric tensor, e.g.  $(\dot{\mathbf{q}} - \mathbf{Q})^T \mathbf{M}(\dot{\mathbf{q}} - \mathbf{Q})$ . In particular, it is possible to automatically compute a sensitivity matrix, analogous to the mass matrix in mechanics, that compensates for scale differences among the components of  $F_{\mathbf{q}}$  (see [31].)

The classical method of Lagrange multipliers [13] solves constrained optimization problems by adding to the gradient of the objective a linear combination of constraint gradients, with unknown coefficients. One then solves simultaneously for the original unknowns, and for the coefficients. In the case of linear constraints and a quadratic objective, this is a linear problem.

The two problems we wish to solve—obtaining  $\dot{\mathbf{p}}^i$  given  $\dot{\mathbf{q}}$ , and  $\dot{\mathbf{q}}$  given  $\dot{\mathbf{p}}^i$ —seek to minimize the same objective subject to the same constraints, differing only in regard to the knowns and unknowns. Even so, the solutions will turn out to be quite different because of the structure of  $C^j$ 's dependencies on  $\dot{\mathbf{p}}^i$  and  $\dot{\mathbf{q}}$ . We next consider each problem in turn.

### 3.3 Floaters

In solving for the  $\dot{\mathbf{p}}^i$ 's, the requirement that the gradient of the objective be a linear combination of the constraint gradients is expressed by

$$G_{\dot{\mathbf{p}}^i} + \sum_j \lambda^j C_{\dot{\mathbf{p}}^i}^j = \dot{\mathbf{p}}^i - \mathbf{P}^i + \lambda^i F_{\mathbf{x}}^i = 0 \quad (4)$$

for some value of the unknown coefficients  $\lambda^i$ . The summation over  $j$  drops out because  $C^j$  cannot depend on  $\dot{\mathbf{p}}^i$  unless  $i = j$ . In addition we require that the constraints be met, i.e. that  $C^i = 0$ ,  $1 \leq i \leq n$ . Equation 4 allows us to express the  $\dot{\mathbf{p}}^i$ 's in terms of the unknown  $\lambda^i$ 's. Substituting for  $\dot{\mathbf{p}}^i$  in equation 3 gives

$$F_{\mathbf{x}}^i \cdot (\mathbf{P}^i - \lambda^i F_{\mathbf{x}}^i) + F_{\mathbf{q}}^i \cdot \dot{\mathbf{q}} + \phi F^i = 0.$$

We may solve for each  $\lambda^i$  independently. Doing so yields

$$\lambda^i = \frac{F_{\mathbf{x}}^i \cdot \mathbf{P}^i + F_{\mathbf{q}}^i \cdot \dot{\mathbf{q}} + \phi F^i}{F_{\mathbf{x}}^i \cdot F_{\mathbf{x}}^i}.$$

Substituting into equation 4 yields

$$\dot{\mathbf{p}}^i = \mathbf{P}^i - \frac{F_{\mathbf{x}}^i \cdot \mathbf{P}^i + F_{\mathbf{q}}^i \cdot \dot{\mathbf{q}} + \phi F^i}{F_{\mathbf{x}}^i \cdot F_{\mathbf{x}}^i} F_{\mathbf{x}}^i \quad (5)$$

which is the particle velocity that solves the constrained optimization problem. Notice that in the case that the surface is not moving and the constraints are met, so that  $F^i = 0$  and  $\dot{\mathbf{q}} = 0$ , this reduces to

$$\dot{\mathbf{p}}^i = \mathbf{P}^i - \frac{F_{\mathbf{x}}^i \cdot \mathbf{P}^i}{F_{\mathbf{x}}^i \cdot F_{\mathbf{x}}^i} F_{\mathbf{x}}^i,$$

which is just the orthogonal projection of  $\mathbf{P}^i$  onto the surface's tangent plane at  $\mathbf{p}^i$ .

### 3.4 Control Points

We follow the same procedure in solving for  $\dot{\mathbf{q}}$ , except that derivatives of  $C^j$  and  $G$  are taken with respect to  $\dot{\mathbf{q}}$ . The condition that the gradient of the objective be a linear combination of the constraint gradients is

$$G_{\dot{\mathbf{q}}} + \sum_j \lambda^j C_{\dot{\mathbf{q}}}^j = \dot{\mathbf{q}} - \mathbf{Q} + \sum_j \lambda^j F_{\mathbf{q}}^j = 0. \quad (6)$$

This time, the sum does not vanish, because every  $C^j$  generally depends on  $\dot{\mathbf{q}}$ .

We next use equation 6 to substitute for  $\dot{\mathbf{q}}$  in equation 3:

$$F_{\mathbf{x}}^i \cdot \dot{\mathbf{p}}^i + F_{\mathbf{q}}^i \cdot \left( \mathbf{Q} - \sum_j \lambda^j F_{\mathbf{q}}^j \right) + \phi F^i = 0.$$

Rearranging gives us the  $n \times n$  matrix equation to be solved for  $\lambda^j$ :

$$\sum_j \left( F_{\mathbf{q}}^i \cdot F_{\mathbf{q}}^j \right) \lambda^j = F_{\mathbf{q}}^i \cdot \mathbf{Q} + F_{\mathbf{x}}^i \cdot \dot{\mathbf{p}}^i + \phi F^i. \quad (7)$$

Note that element  $(i, j)$  of the matrix is just the dot product  $F_{\mathbf{q}}^i \cdot F_{\mathbf{q}}^j$ . Having solved for the  $\lambda^j$ 's, we then solve for  $\dot{\mathbf{q}}$  using equation 6:

$$\dot{\mathbf{q}} = \mathbf{Q} - \sum_j \lambda^j F_{\mathbf{q}}^j. \quad (8)$$

### 3.5 Summary

In this section we have given the solutions to two very closely related problems:

- Given the instantaneous surface motion  $\dot{\mathbf{q}}$ , solve for particle velocities  $\dot{\mathbf{p}}^i$  that minimize deviation from desired velocities  $\mathbf{P}^i$  subject to the constraint that the particles stay on the surface. Each particle's constrained velocity may be computed independently.
- Given the particle velocities  $\dot{\mathbf{p}}^i$ , solve for the implicit function time derivative  $\dot{\mathbf{q}}$  that minimizes deviation from a desired time derivative  $\mathbf{Q}$ , again, subject to the constraint that the particles must remain on the surface. Calculating  $\dot{\mathbf{q}}$  entails the solution of an  $n \times n$  linear system, where  $n$  is the number of particles.

We combine these methods by maintaining two populations of particles: *control points* and *floaters*. Control points are moved explicitly by the user, and  $\dot{\mathbf{q}}$  is calculated to make the surface follow them. In contrast, floaters' velocities are calculated to make them follow the surface, once  $\dot{\mathbf{q}}$  has been computed.

## 4 Adaptive Sampling

In this section we address the problem of sampling implicit surfaces, building on the floater mechanism that we presented in the previous section. Good sampling is a requirement both for quick rendering and for the evaluation of integrals such as surface area or volume.

Our primary goal is to obtain sampling distributions that are either (a) uniform, with user-specified density, (b) or non-uniform, with density based on local criteria such as surface curvature. We wish to reach the specified distribution quickly from a few seed points (ideally, only one per connected component) and to *maintain* a good distribution as the surface moves and deforms. To support interactive sculpting, we must be able to update at least a few hundred sample points at 10Hz or better. Additional goals are that the particles should move as little as possible in response to surface motion, and that only basic and generic information about the function  $F$  be required. It should not be necessary to supply a surface parameterization.

The starting point for our approach is the idea, introduced by Turk [28] and by Figueiredo *et al.* [12], that particles can be made to spread out to uniform density by local repulsion, relying on the finiteness of the surface to limit growth. Simple repulsion can do a good job at ironing out local irregularities given a reasonably good initial sampling (as in Turk's application to resampling of a polygon

mesh) but is extremely slow to converge if the initial sampling is irregular at large scale, and fails completely to track surface motions and deformations.

After describing our basic repulsion scheme, we introduce the idea of *global fissioning*: we start the sampling process with a very small number of particles but a very large radius of interaction, coming close to equilibrium in just a few iterations. We then fission each particle, imposing random displacements that are smaller than the interaction radius. At the same time, we scale the interaction radius to a smaller value. We now have a new starting point, locally irregular but with nearly uniform large-scale structure. A few iterations suffice to smooth out the small irregularities and reach a new equilibrium. The scaling and fissioning process is repeated until the target sampling density is reached.

Global fissioning still fails to handle surface motion: should new nonuniformities be introduced after the fissioning process terminates, the system suffers all of the shortcomings of simple fixed-scale repulsion. So, for example, the sudden introduction of a bulge in the surface can create a gaping hole in the sampling pattern that will be repaired extremely slowly, if at all. Intuitively, we would like particles at the edge of such voids to "feel" the reduction of density, expand their radii of interaction to quickly fill the hole, then begin fissioning to restore full density. On the other hand, if density becomes too high, we would like particles to die off until the desired density is restored. We will conclude the section by describing a fast and robust adaptive repulsion scheme that provides just this behavior, meeting all of our goals.

### 4.1 Simple Repulsion

As a windowed density measure, we employ a simple Gaussian energy function based on distances between particles in 3-D. We define the *energy* of particle  $i$  due to particle  $j$  to be:

$$E^{ij} = \alpha \exp\left(-\frac{|\mathbf{r}^{ij}|^2}{2\sigma^2}\right)$$

where  $\mathbf{r}^{ij} = \mathbf{p}^i - \mathbf{p}^j$  is the vector between particles,  $\alpha$  is a global repulsion amplitude parameter, and  $\sigma$ , called the *global repulsion radius*, is the standard deviation of the Gaussian. The repulsion radius controls the range of the repulsion "force." Note that  $E^{ij} = E^{ji}$ .

The energy of particle  $i$  in its current position is defined as:

$$E^i = \sum_{j=1}^n E^{ij}$$

Ultimately, we would like to reach the global minimum of each  $E^i$  by varying the particle positions on the surface. Finding the global minimum is impractical, but we can find a local minimum by gradient descent: each particle moves in the direction that reduces its energy fastest. We therefore choose each particle's desired velocity to be negatively proportional to the gradient of energy with respect to its position:

$$\mathbf{P}^i = -\sigma^2 E_{\mathbf{p}^i}^i = -\sum_{j=1}^n \mathbf{r}^{ij} E^{ij}$$

The formulas for energy and desired velocity have been carefully chosen here so that "energy" is unitless, while desired velocity is proportional to distance. This guarantees that the sampling pattern computed by this simple repulsion method scales with a surface.

If desired particle velocities are set in this way, and constrained particle velocities are computed with equation 5, particles repel, but their behavior is highly dependent on the parameter  $\sigma$ . The slope of a

Gaussian peaks at distances of  $\pm\sigma$  and it is near zero at much smaller or much greater distances. When the distance between particles is not between  $.03\sigma$  and  $3\sigma$ , for instance, the repulsion is below 7% of its peak. If  $\sigma$  is chosen too small then particles will (nearly) stop spreading when their separation is about  $3\sigma$ , and if  $\sigma$  is chosen too big then distant particles will repel more than nearby ones, and the resulting sampling pattern will be poor. The best value for  $\sigma$  is about  $.3\sqrt{(\text{surface area})/(\text{number of particles})}$ .

## 4.2 Global Fissioning

If a surface is seeded with several floater particles, and an initial value of  $\sigma$  can be found that causes these particles to disperse, then the sampling can be repeatedly refined by allowing the particles to reach equilibrium, then simultaneously fissioning each particle into two, giving the new particles a small random displacement, and simultaneously dividing  $\sigma$  by  $\sqrt{2}$ . The particles are considered to be at equilibrium when their net forces, and hence their speeds, get low. With this global fissioning scheme, early generations will spread out sparsely, and succeeding generations will fill in more densely.

Simple repulsion with global fissioning is acceptable for maintaining a good distribution on a very slowly changing surface, but the population is always a power of two, and particles do not redistribute quickly in response to rapid surface changes. Global fissioning fails to adapt to changes in a surface adequately, as mentioned earlier.

## 4.3 Adaptive Repulsion

To develop a more adaptive repulsion scheme, we employ an analogy to a population of organisms distributing itself uniformly across an area. Specifically, imagine a population of pioneers spreading West and colonizing America. In order to settle the entire country as quickly as possible, a good rule is for each male-female pair to spread out as much as possible away from their neighbors, until the encroachment on them is roughly equal in all directions, and only then to homestead and have children. If the encroachment from neighbors is low, then each pair can claim more land (be greedier), but when neighbors are pressing in, each pair must relinquish land. Early pioneers travel great distances and claim huge tracts of land, while later generations move less and divide up successively smaller shares until the desired density is achieved.

These ideas can be applied to particle behavior. To achieve uniform densities quickly, and maintain them as the surface moves or deforms, we will allow each particle to have its own repulsion radius  $\sigma^i$ , and to decide independently when it should fission or die. A particle's radius should grow when all of the forces on it are small and it should shrink when the forces on it are big. For a particle near equilibrium, birth and death occur when the density is too low or too high, respectively. We now quantify these principles.

Similar to the simple repulsion scheme, we define the energy of particle  $i$  due to particle  $j$  as:

$$E^{ij} = \alpha \exp\left(-\frac{|\mathbf{r}^{ij}|^2}{2(\sigma^i)^2}\right)$$

Note that the global parameter  $\sigma$  has been replaced by the local parameter  $\sigma^i$ , so that  $E^{ij} \neq E^{ji}$  in general.

The energy at particle  $i$  is defined as:

$$E^i = \sum_{j=1}^n (E^{ij} + E^{ji})$$

The repulsion force and desired velocity is again proportional to the

gradient of energy with respect to position:

$$\mathbf{P}^i = -(\sigma^i)^2 E_{\mathbf{p}^i}^i = (\sigma^i)^2 \sum_{j=1}^n \left( \frac{\mathbf{r}^{ij}}{(\sigma^i)^2} E^{ij} - \frac{\mathbf{r}^{ji}}{(\sigma^j)^2} E^{ji} \right) \quad (9)$$

The time-varying repulsion radii will be controlled differentially. We want the radius to grow when the energy is too low and to shrink when the energy is too high. This can be done indirectly by controlling the energies.

As stated earlier, our energy measure is scale-invariant. That is, if all surfaces and samples are scaled ( $\mathbf{p}^i$  and  $\sigma^i$ ), the  $E^i$  will remain constant. Therefore, to ensure that neighboring particles repel each other, we can simply drive all of their energies to a global desired energy level,  $\hat{E}$ . To arrive at a value for  $\hat{E}$ , we consider an ideal hexagonal close-packing, which is the best uniform sampling pattern for a planar surface. In this configuration, all  $\sigma^i$  should be equal, and the distance between nearest neighbors should be roughly  $2\sigma$  to guarantee strong repulsion forces. Since each particle has six nearest neighbors in this configuration, the desired energy should be roughly  $\hat{E} = 6\alpha \exp(-(2\sigma)^2/(2\sigma^2)) = 6e^{-2}\alpha \approx .8\alpha$ .

The portion of a particle's repulsion energy that is directly affected by a change in its own repulsion radius is:

$$D^i = \sum_{j=1}^n E^{ij}$$

To keep  $D^i$  near the desired value, we use the linear feedback equation:

$$\dot{D}^i = -\rho(D^i - \hat{E}) \quad (10)$$

where  $\rho$  is the feedback constant.

The change to the repulsion radius of a particle that will yield this change in energy can be derived with the chain rule:  $\dot{D}^i = D_{\sigma^i}^i \dot{\sigma}^i + \sum_j D_{\mathbf{p}^j}^i \cdot \dot{\mathbf{p}}^j$ , neglecting the latter terms, thus:

$$\dot{\sigma}^i = \frac{\dot{D}^i}{D_{\sigma^i}^i} \quad (11)$$

The rule above works fine for particles that are exerting some force on their neighbors, but it causes infinite radius change when a particle is alone in a sparsely sampled region of a surface (or is the first particle), where  $D^i = D_{\sigma^i}^i = 0$ . In such cases we want the radius to grow, but not catastrophically, so we modify equation 11:

$$\dot{\sigma}^i = \frac{\dot{D}^i}{D_{\sigma^i}^i + \beta} \quad (12)$$

for some  $\beta$ . The change in energy with respect to a change in radius is:

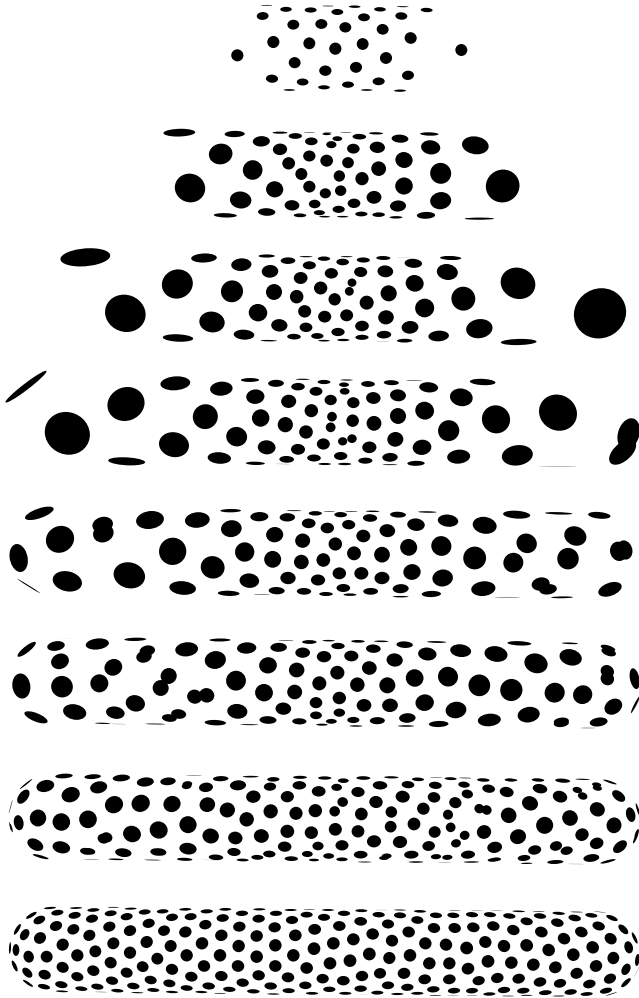
$$D_{\sigma^i}^i = \frac{1}{(\sigma^i)^3} \sum_{j=1}^n |\mathbf{r}^{ij}|^2 E^{ij} \quad (13)$$

Using equations 9, 12, 10, and 13 to control particle positions and repulsion radii will do a good job of moving particles into sparse regions quickly, but their radii might become very large, and hence the density might remain too low.

## 4.4 Adaptive Fission/Death

To achieve uniform density it is necessary that large-radius particles fission. Likewise, particles that are overcrowded should be considered for death.

We use the following criteria to control birth and death of particles: A particle is fissioned iff:



**Figure 1:** This sequence illustrates the adaptive repulsion and fissioning mechanism. The topmost image shows a deliberately poor sampling of a blobby cylinder produced using simple repulsion: the cylinder was rapidly stretched, leaving the sample points behind. The remaining images, from top to bottom, show the recovery of good sampling when adaptive repulsion is enabled. The particles at the frontier increase their radii of repulsion, rapidly filling the voids. As the particles slow down, they fission, restoring the desired sampling density. This process takes about four seconds on an SGI Crimson.

- the particle is near equilibrium,  $|\dot{\mathbf{p}}^i| < \gamma\sigma^i$ , and
- either the particle's repulsion radius is huge ( $\sigma^i > \sigma^{\max}$ ), or it is adequately energized and its radius is above the desired radius ( $D^i > v\hat{E}$  and  $\sigma^i > \hat{\sigma}$ ).

Fission splits a single particle in two. The two particles are given initial radii of  $\sigma^i/\sqrt{2}$  and a desired velocity that is a random direction scaled by a fraction of  $\sigma^i$ . A particle dies iff:

- the particle is near equilibrium,  $|\dot{\mathbf{p}}^i| < \gamma\sigma^i$ , and
- the particle's repulsion radius is too small,  $\sigma^i < \delta\hat{\sigma}$ , and
- the following biased randomized test succeeds:  $R > \sigma^i/(\delta\hat{\sigma})$ , where  $R$  is a uniform random number between 0 and 1.

The death criteria are made stochastic to prevent mass suicide in overcrowded regions.

This combination of adaptive repulsion, fissioning, and death is much more responsive to changes in the surface shape than the simple repulsion scheme.

## 5 Implementation and Results

The techniques described above have been implemented in about 3700 lines of C++ code. Particular implicit function classes are derived from a generic implicit function base class. Adding a new implicit function to the system is easy, requiring only the implementation of functions  $F$ ,  $F_{\mathbf{x}}$ ,  $F_{\mathbf{q}}$ , and bounding box. Each of these except  $F_{\mathbf{q}}$  is standard in any system employing implicit functions.

For example, we define the blobby sphere implicit function to be the sum of Gaussians of the distance to each of  $k$  center points [8]. The parameter vector  $\mathbf{q}$  consists of  $4k + 1$  parameters: a bias  $b$  plus four parameters for each sphere (a center 3-vector  $\mathbf{c}^i$  and standard deviation  $s^i$ ). Thus,

$$\mathbf{q} = [b, \mathbf{c}^1, s^1, \mathbf{c}^2, s^2, \dots, \mathbf{c}^k, s^k]$$

If we define

$$g^i(\mathbf{x}) = \exp\left(-\frac{|\mathbf{x} - \mathbf{c}^i|^2}{(s^i)^2}\right)$$

then the functions needed by the system are

$$F(\mathbf{x}) = b - \sum_{i=1}^k g^i(\mathbf{x})$$

$$F_{\mathbf{x}}(\mathbf{x}) = 2 \sum_i \frac{\mathbf{x} - \mathbf{c}^i}{(s^i)^2} g^i(\mathbf{x})$$

$$F_{\mathbf{q}}(\mathbf{x}) = [F_b, F_{\mathbf{c}^1}, F_{s^1}, F_{\mathbf{c}^2}, F_{s^2}, \dots, F_{\mathbf{c}^k}, F_{s^k}]$$

where

$$F_b(\mathbf{x}) = 1$$

$$F_{\mathbf{c}^i}(\mathbf{x}) = -2 \frac{\mathbf{x} - \mathbf{c}^i}{(s^i)^2} g^i(\mathbf{x})$$

$$F_{s^i}(\mathbf{x}) = -2 \frac{|\mathbf{x} - \mathbf{c}^i|^2}{(s^i)^3} g^i(\mathbf{x})$$

If we assume that  $g^i(\mathbf{x}) = 0$  beyond a radius of  $3s^i$ , then a conservative bounding box for blobby spheres is the bounding box of non-blobby spheres with centers  $\mathbf{c}^i$  and radii  $3s^i$ .

We have also implemented spheres and blobby cylinders. A blobby cylinder function is defined to be the sum of Gaussians of the distance to each of several line segments. A system of  $k$  blobby cylinders has  $7k + 1$  parameters: a bias plus seven parameters for each cylinder (two endpoints and a standard deviation).

It is often useful to freeze some of these parameters to a fixed value so that they will not be modified during interaction. This is

done simply by leaving them out of the  $\mathbf{q}$  and  $F\mathbf{q}$  vectors. To get blobs of equal radii, for instance, one would omit all  $s^i$ .

The system starts up with a single floater positioned arbitrarily in the bounding box of the surface and then begins the physical simulation by repeating the following differential step:

- The user interface sets *desired* control point velocities  $\mathbf{P}^i$ . Stationary control points of course have zero desired velocity, while control points being dragged by the user have desired velocities that are calculated as a function of cursor position.
- Set  $\mathbf{Q}$ , the *desired* values for the time derivatives of the surface parameters. These are typically set to zero to minimize parametric change in the surface, but they could also be calculated to attract the surface toward a default shape.
- Compute the actual surface parameter changes,  $\dot{\mathbf{q}}$ , as constrained by the control point velocities, using equations 7 and 8.
- Compute repulsion forces between floaters to set their desired velocities  $\mathbf{P}^i$ , using equation 9.
- Compute actual floater velocities, as constrained by the already-computed surface time derivatives, using equation 5. (When the gradient  $F_{\mathbf{x}}$  is near zero, however, the surface is locally ill-defined, and it is best to leave such floaters motionless, i.e.,  $\dot{\mathbf{p}}^i = 0$ .)
- Compute the change to floater repulsion radii,  $\dot{\sigma}^i$ , using equations 12, 10, and 13.
- Update the positions of the control points and floaters using Euler's method, that is:  $\mathbf{p}^i(t + \Delta t) = \mathbf{p}^i(t) + \Delta t \dot{\mathbf{p}}^i(t)$ , and similar formulas to update the surface parameters  $\mathbf{q}$  from  $\dot{\mathbf{q}}$ , and the floater repulsion radii  $\sigma^i$  from  $\dot{\sigma}^i$ .
- Test each floater for possible fission/death.
- Redisplay the floaters and control points as disks tangent to the surface, with normal given by  $F_{\mathbf{x}}$  and (for floaters) radius proportional to  $\sigma^i$ .

Using the mouse, the user can pick a control point and move it in a plane perpendicular to the view direction. Pulling control point  $i$  sets the desired control point velocity  $\mathbf{P}^i$ . Since the velocities feed into the constrained optimization solution, which in turn feeds into a numerical differential equation solver, some care must be taken to ensure that control point motions are reasonably smooth and well behaved, which they might not be if positions were set directly by polling the pointing device. A simple solution which works well is to make the velocity of the dragged particle proportional to the vector from the point to the 3-D cursor position. This in effect provides spring coupling between the cursor and the control point. Although the control point can lag behind the cursor as a result, performance is brisk enough that the lag is barely noticeable. Similar dragging schemes are described in [14,31]. The user can also create and delete control points and adjust the desired repulsion radius  $\hat{\sigma}$  through a slider.

The matrix in equation 7 is symmetric and in general positive definite. It thus lends itself to solution by Cholesky decomposition [23], which is easy to implement, robust and efficient. However, the matrix can become singular if inconsistent or redundant constraints are applied, that is if the number of constraints exceeds  $m$ , or if some of the  $F_{\mathbf{q}}^i$ 's are linear combinations of others. While the former condition is easy to detect by counting, the latter is not. The problem of singularities can be circumvented by using a least-squares technique, or singular value decomposition [23].

The system is fast enough to run at interactive rates. Let  $m$  be the number of degrees of freedom of the implicit surface, let  $n$  be the number of control points, and let  $r$  be the number of floaters. The most expensive parts of the algorithm are the computation of the  $n \times n$  matrix of equation 7, which has cost  $O(mn^2)$ , the solution

of the linear system, which has cost  $O(n^3)$ , the computation of repulsion forces between all pairs of floaters, which currently has cost  $O(r^2)$ , and the display of the floaters, which has cost  $O(r)$  (with a large constant). Our current system does not handle overconstrained surfaces, so  $m \geq n$ , thus the total asymptotic cost of the algorithm is  $O(mn^2 + r^2)$  per iteration.

We have run simulations as complex as  $m = 56, n = 10, r = 500$ . Above  $r = 250$  floaters, the  $O(r^2)$  repulsion cost has dominated, but this could easily be optimized using spatial data structures. For smaller numbers of floaters ( $r < 150$ ), our system runs at interactive rates (10 Hz or faster on a Silicon Graphics workstation with 100 MHz processor).

The following parameter settings are recommended (where  $d$  is surface diameter):

PARAMETER	MEANING
$\Delta t = .03$	time step
$\phi = \rho = 15$	feedback coefficients to keep particles from drifting off surface, and keep particles energized, respectively
$\alpha = 6$	repulsion amplitude
$\bar{E} = .8\alpha$	desired energy
$\beta = 10$	to prevent divide-by-zero
$\hat{\sigma} = d/4$ or less	desired repulsion radius (user-controllable)
$\sigma^{\max} = \max(\frac{d}{2}, 1.5\hat{\sigma})$	maximum repulsion radius (note that this changes over time)
$\gamma = 4$	equilibrium speed (multiple of $\sigma^i$ )
$\nu = .2$	fraction of $\bar{E}$ , for fissioning
$\delta = .7$	fraction of $\hat{\sigma}$ , for death

Most of these parameters can be set once and forgotten. The only parameter that a user would typically need to control is the desired repulsion radius,  $\hat{\sigma}$ .

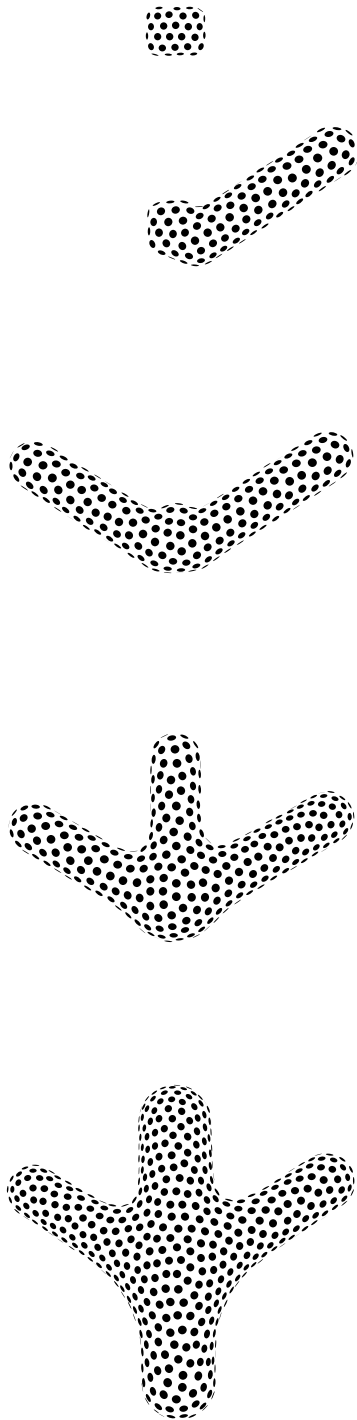
Overall, the method meets our goals, it is fast, and it has proven to be very robust. It has recovered from even violent user interaction causing very rapid shape change. The adaptive sampling, fission, and death techniques seem to be well tuned and to work well together, as we have not seen the system oscillate, diverge, or die with the current parameter settings. During interaction,  $\hat{\sigma}$  is the only parameter that needs to be varied.

Another result of this work is that we have discovered that *implicit surfaces are slippery*: when you attempt to move them using control points they often slip out of your grasp.

## 6 Conclusions

In this paper we have presented a new particle-based method for sampling and control of implicit surfaces. It is capable of supporting real-time rendering and direct manipulation of surfaces. Our control method is not limited to algebraic surfaces as many previous techniques are; it allows fast control of general procedural implicit functions through control points on the surface. We have presented a dynamic sampling and rendering method for implicit surfaces that samples a changing surface more quickly than existing methods. The use of constraint methods allows particles to follow the surface as it changes, and to do this more rapidly and accurately than with penalty methods. Our algorithms for adaptive repulsion, fission, and death of particles are capable of generating good sampling patterns much more quickly than earlier repulsion schemes, and they sample the surface well even during rapid shape changes.





**Figure 2:** This sequence illustrates the construction of a shape composed of blobby cylinders. The shape was created by direct manipulation of control points using the mouse. In the topmost image, all three cylinder primitives are superimposed. Each subsequent image represents the result of a single mouse motion.

There are a number of directions for future research.

We intend to investigate other uses for the samplings we obtain. One of these is the calculation of surface integrals for area, volume, or surface fairness measures such as those described in [19,30]. Another is the creation of polygon meshes.

To polygonize a surface within the framework presented here it is necessary to infer topology from the sample points. This is more difficult than finding a polygonization from a set of samples on a grid in 3-D, as in marching cubes algorithms, where an approximate topology is suggested by the signs of the samples and by the topology of the grid itself. Delaunay triangulation in 2-D or 3-D is one possible way to extract topology [12,27]. A more robust alternative would employ Lipschitz conditions and interval arithmetic [26]. To preserve the basic advantages of our method, we would require a polygonization algorithm that allows efficient dynamic updates as the surface changes.

Although we developed it to sample implicit surfaces, our adaptive repulsion scheme can be applied to meshing or sampling of parametric surfaces as well: each floater would be defined by its position in the surface's 2-D parameter space, rather than position in 3-D space.

Several performance and numerical issues remain to be addressed. As we tackle more complex models, we could exploit sparsity in  $F$ 's dependence on  $\mathbf{q}$ . Notably, with local bases such as blobby models, the dependence of  $F$  on faraway elements is negligible. An additional numerical issue is the handling of singular constraint matrices, due to overdetermined or dependent constraints. Excellent results can be obtained using least-squares techniques.

An additional area of investigation is the use of local criteria, notably surface curvature, to control sampling density. Surface curvature can be measured directly, at the cost of taking additional derivatives of  $F$ . Since this places a considerable extra burden on the implementor of implicit primitives, an alternative is to estimate curvature at each floater based on positions and normals of nearby points. Having established a desired density at each point, based on curvature or any other criterion, relatively simple modifications to the adaptive repulsion scheme will yield the desired nonuniform density. Another possible density criterion is the user's focus of interest, e.g. the neighborhood of a control point being dragged.

Finally, there is room for considerable further work in interactive sculpting of implicit surfaces. Dragging one control point at a time can be somewhat limiting given the slippery behavior of the surface. However, the basic control-point machinery developed here could be used to build more complex sculpting tools that influence multiple surface points in coordinated ways.

## Acknowledgements

The authors wish to thank Scott Draves and Sebastian Grassia for their contributions to this work. This research was supported in part by a Science and Technology Center Grant from the National Science Foundation, #BIR-8920118, by an NSF High Performance Computing and Communications Grant, #BIR-9217091, by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University, by Apple Computer, Inc, and by an equipment grant from Silicon Graphics, Inc. The second author was supported by NSF Young Investigator Award #CCR-9357763.

## References

- [1] Chandrajit Bajaj, Insung Ihm, and Joe Warren. Higher-order interpolation and least-squares approximation using implicit

- algebraic surfaces. *ACM Trans. on Graphics*, 12(4):327–347, Oct. 1993.
- [2] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics*, 23(3):223–232, July 1989.
  - [3] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, August 1990.
  - [4] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. *Computer Graphics*, 26(2):303–308, 1992. Proc. Siggraph '92.
  - [5] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22:179–188, 1988.
  - [6] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics*, 1972.
  - [7] Thaddeus Beier. Practical uses for implicit surfaces in animation. In *Modeling, Visualizing, and Animating Implicit Surfaces (SIGGRAPH '93 Course Notes)*, pages 20.1–20.10, 1993.
  - [8] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. on Graphics*, 1(3):235–256, July 1982.
  - [9] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.
  - [10] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–350. Academic Press, Boston, 1994.
  - [11] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics (1990 Symp. on Interactive 3D Graphics)*, 24(2):109–116, 1990.
  - [12] Luiz Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulos, and Luiz Velho. Physically-based methods for polygonization of implicit surfaces. In *Graphics Interface '92*, pages 250–257, May 1992.
  - [13] Phillip Gill, Walter Murray, and Margret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.
  - [14] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. *Computer Graphics*, 26(2):331–340, 1992. Proc. Siggraph '92.
  - [15] Herbert Goldstein. *Classical Mechanics*. Addison Wesley, Reading, MA, 1950.
  - [16] Huges Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH 93 Proceedings*, pages 19–26, July 1993.
  - [17] David J. Jevans, Brian Wyvill, and Geoff Wyvill. Speeding up 3-D animation for simulation. In *Proc. MAPCON IV (Multi and Array Processors)*, pages 94–100, Jan. 1988.
  - [18] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–170, July 1987.
  - [19] Henry Moreton and Carlo Séquin. Functional minimization for fair surface design. *Computer Graphics*, 26(2):167–176, 1992. Proc. Siggraph '92.
  - [20] Shigeru Muraki. Volumetric shape description of range data using “blobby model”. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):227–235, July 1991.
  - [21] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *Computer Graphics and Applications*, pages 33–41, Nov. 1993.
  - [22] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):145–152, July 1987.
  - [23] W.H. Press, B.P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1988.
  - [24] A. Ricci. A constructive geometry for computer graphics. *Computer Journal*, 16(2):157–160, May 1973.
  - [25] T. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2(1-3):53–60, 1985.
  - [26] John M. Snyder. *Generative Modeling for Computer Graphics and CAD*. Academic Press, Boston, 1992.
  - [27] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):185–194, July 1992.
  - [28] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):289–298, July 1991.
  - [29] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55–64, July 1992.
  - [30] William Welch and Andrew Witkin. Variational surface modeling. *Computer Graphics*, 26(2):157–166, 1992. Proc. Siggraph '92.
  - [31] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. Proc. 1990 Symposium on 3-D Interactive Graphics.
  - [32] Andrew Witkin and William Welch. Fast animation and control of non-rigid structures. *Computer Graphics*, 24(4):243–252, July 1990. Proc. Siggraph '90.
  - [33] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

# Using Particles to Sample and Control More Complex Implicit Surfaces

John C. Hart, Ed Bachta, Wojciech Jarosz, Terry Fleury

University of Illinois

{jch|bachta|wjarosz|tfleury}@uiuc.edu

## Abstract

*In 1994, Witkin and Heckbert developed a method for interactively modeling implicit surfaces by simultaneously constraining a particle system to lie on an implicit surface and vice-versa. This interface was demonstrated to be effective and easy to use on example models containing a few blobby spheres and cylinders. This system becomes much more difficult to implement and operate on more complex implicit models. The derivatives needed for the particle system behavior can become laborious and error-prone when implemented for more complex models. We have developed, implemented and tested techniques for automatic and numerical differentiation of the implicit surface function. Complex models also require a large number of parameters, and the management and control of these parameters is often not intuitive. We have developed adapters, which are special shape-transformation operators that automatically adjust the underlying parameters to yield the same effect as the transformation. These new techniques allow constrained particle systems to sample and control more complex models than before possible.*

## 1 Introduction

Witkin and Heckbert [14] revolutionized implicit surface modeling by using a particle system to both display and control an implicit surface. Their treatment used a real function  $F : \mathbf{R}^3 \times Q \rightarrow \mathbf{R}$  over model space  $\mathbf{R}^3$  and a continuous parameter space  $Q$ . This real function yielded an implicit surface as the solution points  $x$  such that  $F(x, q) = 0$  for a fixed, given vector of parameters  $q \in Q$ .

A particle  $p^i$  constrained to the implicit surface of  $F$  such that  $F(p^i, q) = 0$  is called a *floater*. This constraint is enforced by setting its original velocity  $\dot{P}^i$  to a legal velocity  $\dot{p}^i$  by subtracting any illegal components normal to the implicit surface

$$\dot{p}^i = \dot{P}^i - \frac{F_x^i \cdot \dot{P}^i + F_q \cdot \dot{q} + \phi F^i}{F_x^i \cdot F_x^i} F_x^i. \quad (1)$$

(Note that here  $\dot{P}$  denotes the desired velocity instead of  $P$  [14].) These illegal components are due to either particle velocities ( $F_x^i \cdot \dot{P}^i$ ) or parameter velocities ( $F_q \cdot \dot{q}$ ) that change the resulting value of  $F$ . Hence we need the derivative of  $F$  with respect to both its embedding  $F_x$  and its parameterization  $F_q$ .

The constrained particle system displayed the implicit surface with a collection of disks centered at the particles oriented according to the surface normal. These oriented disks provide a usable and highly responsive display of the underlying implicit surface, and also yield a quasi-volumetric display of the surface that reveals interior structure in the gaps between disks. The visual edge noise created by the disks can sometimes be distracting, but this can be overcome by connecting the particles into a polygonization using a topological guarantee [12].

Some floater particles can be selected as control particles, which means the implicit surface is constrained to pass through these particles. Control particles can be dragged to new locations, and the implicit surface deforms to accommodate its new position. This deformation occurs by changing the parameters  $q$  of the implicit surface using the parameter velocity

$$\dot{q} = \dot{Q} - \sum_j \lambda^j F_q^j. \quad (2)$$

The index  $j$  is the index of the control particle, and  $\dot{Q}$  is the desired unconstrained parameter velocity. The  $\lambda^j$  are Lagrange multipliers found by solving the system

$$\sum_j (F_q^i \cdot F_q^j) \lambda^j = F_q^i \cdot \dot{Q} + F_x^i \cdot \dot{p}^i + \phi F^i. \quad (3)$$

The desired parameter velocity  $\dot{Q}$  is usually zero, such that (2) and (3) dissipate control particle velocities  $\dot{p}$  into parameter velocities  $\dot{q}$ .

Witkin and Heckbert [14] alluded to an object-oriented implicit surface class hierarchy, where new implicit model objects need to implement  $F$ ,  $F_x$ ,  $F_q$  and a bounding box (to keep the particles from following a non-compact manifold indefinitely). Most implicit modeling systems implement the function, its gradient and a bounding box, so the only

new information needed is the derivative of the implicit surface function with respect to its parameters.

The goal of the work described in this paper is to integrate the particle system into a full-featured implicit surface modeling system. Section 2 reviews some previous implicit surface modeling systems. To our knowledge, no one has yet implemented such an object hierarchy in a full-featured implicit modeling system based on the control particle interface.

We have developed such an object-oriented class hierarchy specifically for inclusion into a particle-based modeling system. Section 3 describes the flexibility of our system that includes a large variety of implicit surface primitives and operators.

Witkin and Heckbert [14] suggest (once  $F_q$  is implemented) the application of particle-based interaction to complex hierarchical implicit surface models is straightforward. In the process of implementing such a system we have found several setbacks, specifically in implementing and debugging the function derivatives  $F_x$  and  $F_q$ , and managing large numbers of parameters for manipulating complex composite implicit surface models, as demonstrated by the 30+ parameters shown in Figures 1 and 2.

One of the challenges of implementing a full-featured implicit surface representation for a particle-based modeler is the development of the derivatives needed. The particle-based modeler requires each primitive and operator to have a derivative with respect to its embedding space and its parameters. Section 4 describes several methods available to ease the programming of these derivatives.

Complex hierarchies of implicit operators and primitives quickly grow to become overwhelming. Section 5 describes methods for modeling that use special operators called adapters that reparameterize a complex implicit model. This reparameterization simplifies the modeling process by providing more intuitive parameters to the user.

## 2 Other Complex Implicit Surface Modeling Systems

This paper develops a modeling system capable of creating complex models of a wide variety of implicit primitives and operators using a particle system for display and control. Several other modeling systems have been previously developed to create complex implicit surface models, but each of these system has been limited to a specific subset of implicit primitives and operations.

Witkin and Heckbert [14] used a prototype implementation of a particle-based implicit surface modeler that proved the concept. Their implementation was never released, and only supported small collections of a few primitives, including blobby spheres and cylinders. Pedersen [8] later released a more robust, flexible and freely available particle

system based on (1) of floater particles for displaying implicit surfaces, but it did not include an implementation of (2) and (3) that provided the control particles needed to interactively change the surface parameters. Stander and Hart [12] later described how to use interval analysis and Morse theory to connect surface constrained floater particles into a polygonization. Their implementation was also a proof-of-concept prototype that was never released. It was limited to axis-aligned Gaussian ellipsoids, and the dynamic mesh that connected the particles into a polygonization was very fragile.

The Blob Tree integrated multiple implicit modeling tools into a single scene description hierarchy for implicit surface modeling [15]. The Blob Tree was based on piecewise-polynomial blobby sphere primitives that blend when placed in proximity to each other. The Blob Tree organized objects into groups and groups can be combined with a smooth surface blend or a creased CSG operation. The blend tree also supported other implicit surface operations, for example non-linear deformations.

The Hyperfun system supported collaborative modeling of implicit surfaces [1]. Hyperfun is based on R-functions which generalize blending between primitives to include non-blended CSG operations. Its components include a scene description hierarchy, a declarative language for defining functions, new file formats for communicating implicit surface descriptions, and integration of the results into common graphics systems such as POV-Ray.

Implicit surfaces are also supported by the VTK toolkit [10, 11]. These implicit surfaces could be used as sources for volumetric “structured-points” data pipelines. These pipelines can be made arbitrarily complex, and have been used to model context geometry to aid in the visualization of acquired volumetric data [9]. Implicit surface sources are integrated into the VTK toolkit by defining the function and its gradient, though the VTK tools typically sample the source into a volume before performing any further processing.

## 3 A General Implicit Object Model

Many previous implicit surface modeling systems have focused only on a subset of the available implicit surface models. Our goal is to design a general full-featured implicit surface system that could support any implicit surface representation or operation.

The `Implicit` class sits at the root of our implicit surface hierarchy, and contains three key member functions. The member function `proc(x)` returns the scalar result  $F(\mathbf{x}, \mathbf{q})$ . The member function `grad(x)` returns the 3-vector result  $F_x(\mathbf{x}, \mathbf{q})$ . The member function `procq(x, dq)` puts the derivative  $F_q(\mathbf{x}, \mathbf{q})$  into the vector `dq`. Each of these functions assumes  $\mathbf{q}$  is constant and held

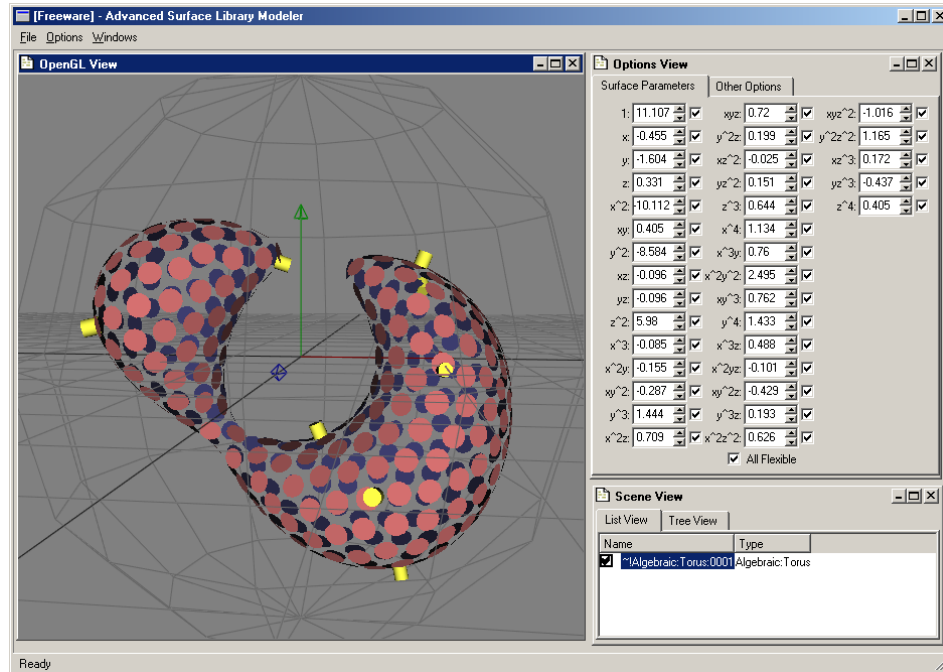


Figure 1. A sample screen of our particle-based modeler directly manipulating a quartic surface. This quartic surface has 35 continuous parameters shown on the right that can be entered individually or selected to be set by the controller particle constraints.

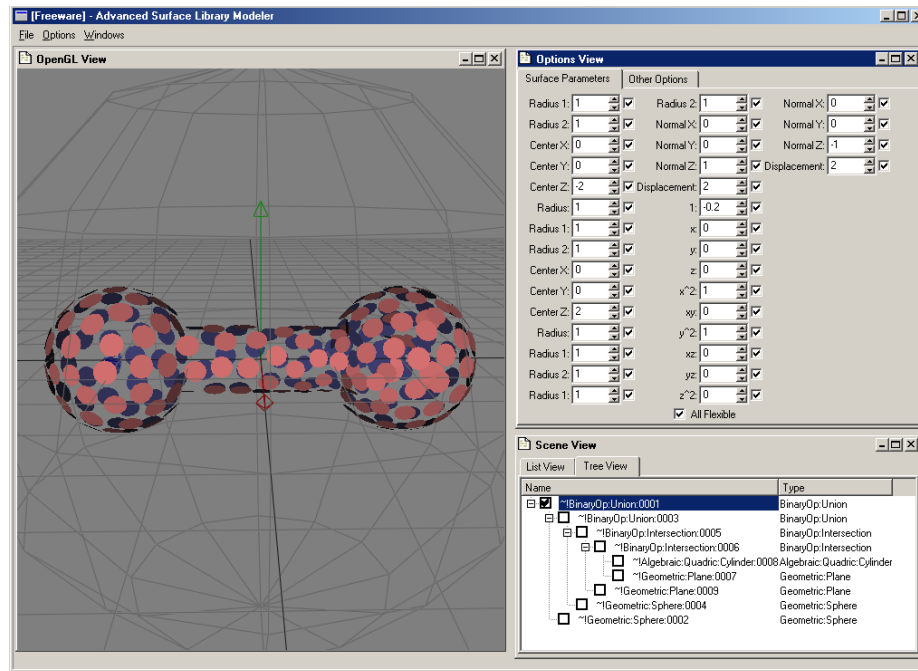


Figure 2. The particle-based modeler directly manipulating a dumbbell modeled using R-functions. This composite surface has 34 continuous parameters shown on the right that can be entered individually or selected to be set by the controller particle constraints.

within the `Implicit` as part of its internal state.

We provide member access to the continuous parameters through an interface uniform across all classes derived from `Implicit`. The member function `getq(q)` puts the current parameters in vector `q`, whereas the member `setq(q)` sets the current parameters to those in vector `q`. This parameter access allows the particle system to query and adjust the continuous parameters as necessary to apply the floater and control particle constraints. The actual implementation of a specific implicit object need not store its parameters in a vector, and can provide additional specialized access methods to its parameters<sup>1</sup>.

We have also implemented implicit surface operators (e.g. offset, scale, union, etc.). These operators affect the inputs and/or the outputs of one or more operand implicit objects. The `getq` and `setq` functions for the operators place the operator's parameters at the beginning of the vector, and follow them with the parameters of its operands. Hence, one can access all of the parameters of a hierarchically-defined implicit object through the `getq/setq` interface of the root operator object in the hierarchy.

## 4 Differentiation

One of the main obstacles in our implementation of a variety of implicit surface models has been in the derivation, implementation and debugging of derivatives of the functions.

### 4.1 Operator-Level Automatic Differentiation

The derivatives of operators, namely `grad` and `procq`, are implemented using the chain rule, eventually calling `grad` and `procq` of the operands. The operations can be considered an arithmetic on implicit objects, which makes the chain-rule `grad` and `procq` implementations a partial form of automatic differentiation,

One could define arithmetic operations as `Implicit` operators. For example, the operator `Times` implements the function  $FG$  as

```
Times::proc(x) {
    return F->proc(x) * G->proc(x);
}
Times::grad(x) {
    return F->grad(x)*G->proc(x) +
           F->proc(x)*G->grad(x);
}
```

<sup>1</sup>For example, we have derived an `Algebraic` class from `Implicit` that provides access to the  $d^3/6 + d^2 + 11d/6 + 1$  coefficients of a degree  $d$  trivariate polynomial. The coefficients are returned sorted by degree, then by  $x, y$  and  $z$  exponents. However, access to these coefficients is much easier by providing the exponents of  $x, y$  and  $z$  than providing a single coefficient index.

Given a sufficient set of these arithmetic operations, one could implement any implicit model. Implicit models constructed by composing these operators would have their differential functions automatically defined, since these objects compute their own derivatives. However, such an implementation would be cumbersome and inefficient.

Rather than implement high-level shape operators using automatically differentiated low-level arithmetic operators, we chose instead to automatically differentiate the high-level shape operators. A good example is our implementation of an abstract `Blend` class. Our blend class assumes the blend combines the isocontours of its operand implicit objects  $F$  and  $G$  [6]. The blend is thus defined by a real bivariate function  $h(f, g)$  of the values  $f, g$  returned by the operands. The blend surface is thus implemented as

```
Blend::proc(x) {
    return h(F->proc(x), G->proc(x));
}
```

Specific blends are derived from the `Blend` class, and define the pure virtual function  $h$ . The blobby model can blend arbitrary implicit surfaces using the function

$$h(f, g) = T - e^{-f-a} - e^{-g-b} \quad (4)$$

where  $T, a$  and  $b$  control the “blobbiness” of the blend [4]. The superelliptical blend is given in this form as

$$h(f, g) = \frac{(f-a)^d}{a^d} + \frac{(g-b)^d}{b^d} - 1 \quad (5)$$

where  $a$  and  $b$  are continuous parameters controlling the extent of the blend and the discrete parameter  $d$  is the degree of the blending function [6]. R-functions provide a continuous neighborhood for CSG operations, and are given by

$$h(f, g) = (f + g + s\sqrt{f^2 + g^2})(f^2 + g^2)^{d/2} \quad (6)$$

where the discrete parameters  $s = \pm 1$  differentiates between union and intersection, and  $d$  again provides a degree of smoothness [7].

Using the chain rule, we define

```
Blend::grad(x) {
    return hf(F->proc(x), G->proc(x))*F->grad(x) +
           hg(F->proc(x), G->proc(x))*G->grad(x);
}
```

using the partial derivative methods `hf` and `hg`. Thus, classes derived from `Blend` need not implement `proc`, `grad` and `procq`, but must implement the simpler method `h` and its partial derivatives `hf` and `hg`.

### 4.2 Code-Level Automatic Differentiation

Given a `proc` implementation, another technique for automatically generating the derivatives `grad` and `procq` is

to apply the automatic differentiation method to the `proc` procedure source code. Computational differentiation is an automatic differentiation applied to algorithms by declaring some variables as dependent and others as independent, and synthesizing the source code necessary to yield the derivatives of the dependent variables with respect to the independent variables. For example, recent tools exist (ADOL-C, ADIC) that differentiate C language source code [5, 3]. Performing automatic differentiation at compile time yields faster derivatives than automatically differentiating at run time.

### 4.3 Numerical Differentiation

We can also use numerical techniques to evaluate the derivatives. Forward differencing of the spatial derivative is implemented as

$$F_{\mathbf{x}}(\mathbf{x}, \mathbf{q}) = \begin{pmatrix} \frac{F(\mathbf{x} + \epsilon \mathbf{e}_0, \mathbf{q}) - F(\mathbf{x}, \mathbf{q})}{\epsilon} \\ \frac{F(\mathbf{x} + \epsilon \mathbf{e}_1, \mathbf{q}) - F(\mathbf{x}, \mathbf{q})}{\epsilon} \\ \frac{F(\mathbf{x} + \epsilon \mathbf{e}_2, \mathbf{q}) - F(\mathbf{x}, \mathbf{q})}{\epsilon} \end{pmatrix} \quad (7)$$

where  $\mathbf{e}_i$  is a unit vector in the  $i$ th dimension direction. Using this notation, the parameter derivative can be similarly derived

$$F_{\mathbf{q}}(\mathbf{x}, \mathbf{q}) = (\dots, F(\mathbf{x}, \mathbf{q} + \epsilon \mathbf{e}_i) - F(\mathbf{x}, \mathbf{q}), \dots). \quad (8)$$

We have found that the constrained particle system remains stable even when numerical versions of  $F_{\mathbf{x}}$  and  $F_{\mathbf{q}}$  are used. The symbolic  $F_{\mathbf{x}}$  runs about four times as fast as the numerical  $F_{\mathbf{x}}$  because the forward differencing implementation calls the implicit surface function four times. Similarly, the symbolic  $F_{\mathbf{q}}$  implementation is  $|\mathbf{q}|$ -times faster than its numerical version.

The virtual methods `grad` and `procq` of our `Implicit` object default to the forward differences approximations. Hence, we can add a new implicit surface model into our library as a black-box by implementing the method `proc`. This task is a less daunting than deriving  $F_{\mathbf{q}}$  by hand, as has been previously suggested [14].

## 5 Parameterization

A second problem with using particles to manipulate complex implicit surface models is the management of the parameters  $\mathbf{q}$  of the implicit surface model. This problem can be decomposed into two specific issues. The first issue is the conceptual disconnection between an object's intuitive parameters (such as location and orientation) and its actual parameters (such as the coefficients of an algebraic). The second issue is that there are often an overwhelmingly large number of free parameters, even for a moderately complex implicit surface model.

One problem we have found is that it is difficult to translate the ellipsoid

$$F(x, y, z, q_0, \dots, q_9) = q_4 x^2 + q_5 xy + q_6 y^2 + q_7 xz + q_8 xy + q_9 z^2 + q_1 x + q_2 y + q_3 z + q_0. \quad (9)$$

using the control particles. We have the ability to select which parameters the control particles affect, but identifying which of the ten quadric coefficients control translation is not intuitive.

In order to translate the ellipsoid by  $\mathbf{o} = (o_x, o_y, o_z)$ , we need to apply the domain transformation  $F(x - o_x, y - o_y, z - o_z, \mathbf{q})$ . Evaluating (9) and collecting terms shows that translation does not affect the parameters  $q_4$  through  $q_9$ , but affects  $q_0$  through  $q_3$  as

$$q_0 \leftarrow q_0 + q_4 o_x^2 + q_5 o_x o_y + q_6 o_y^2 + q_7 o_x o_z + q_8 o_y o_z + q_9 o_z^2 + q_1 o_x + q_2 o_y + q_3 o_z, \quad (10)$$

$$q_1 \leftarrow q_1 - 2q_4 o_x - q_5 o_y - q_7 o_z, \quad (11)$$

$$q_2 \leftarrow q_2 - q_5 o_x - 2q_6 o_y - q_8 o_z, \quad (12)$$

$$q_3 \leftarrow q_3 - q_7 o_x - q_8 o_y - 2q_9 o_z. \quad (13)$$

Enabling only these four coefficients to be changed by the control particles actually causes the entire ellipsoid to deform instead of translate because (2) and (3) dissipate particle velocity evenly among the parameter velocities. The velocities of the ellipsoid parameters due to translation are in fact  $\dot{\mathbf{q}} = (q_1 o_x + q_2 o_y + q_3 o_z + q_4 o_x^2 + q_5 o_x o_y + q_6 o_y^2 + q_7 o_x o_z + q_8 o_y o_z + q_9 o_z^2, -2q_4 o_x - q_5 o_y - q_7 o_z, -q_5 o_x - 2q_6 o_y - q_8 o_z, -q_7 o_x - q_8 o_y - 2q_9 o_z, 0, 0, 0, 0, 0, 0)$ .

### 5.1 Adapters

We solve this problem with the construction of a special kind of operator called an adapter. Whereas operators are designed to remain in the model, adapters are temporary and are used to control the parameters of a model during interactive editing.

An operator creates its parameter vector from its parameters and the parameters of its operands. This assumes that the operator's parameters are independent of the operands' parameters (e.g. the radius of an offsetting operation). The parameters of an adapter are assumed to be related to a subset of the parameters of its operands. The parameter vector of the adapter contains only the parameters of the adapter, and ignores the parameters of the adapter's operands.

For example, the adapter `Mover` is defined by

```
Mover::proc(x) { return F->proc(x - o) }
```

where  $F$  is the operand of `Mover` and  $\mathbf{o}$  is an offset vector. The parameters specific to `Mover` consist only of the

offset vector. If `Mover` was an ordinary operand, then its parameter vector  $\mathbf{q}$  would be  $\mathbf{o}$  catenated with whatever parameters operand object  $F$  may have. Assuming  $F$  has been sufficiently parameterized, the additional components to  $\mathbf{q}$  offered by the offset vector  $\mathbf{o}$  would be redundant.

Since `Mover` is an adapter, we mask all of its operand's parameters, such that `Mover::procq()` returns only  $\mathbf{o}$ . This restricted parameter vector allows the constrained particle system to move an object using a single control particle. But in order for the object to remain in its new location, the adapter must remain attached. One can imagine a model becoming quite complex with adapters every time a subset of the model needs to be positioned.

We can remove an adapter if its parameters are set to its identity configuration, (zeroed in the case of `Mover`). Hence we need a way of transferring changes in parameters in the adapter to parameters in the adapter's operand.

An adapter implements some deformation function  $D : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ . We can apply the deformation  $D$  to the implicit surface of  $F(\mathbf{x}, \mathbf{q}_0)$  as a domain transformation, yielding the implicit surface of  $F(D^{-1}(\mathbf{x}), \mathbf{q}_0)$ . We need to find a new set of parameters  $\mathbf{q}_1$  such that

$$F(D^{-1}(\mathbf{x}), \mathbf{q}_0) = F(\mathbf{x}, \mathbf{q}_1) \quad \forall \mathbf{x} \in \mathbf{R}^3. \quad (14)$$

The adapter function  $D$  has its own set of parameters  $\mathbf{q}_D$ . (In the `Mover` example,  $\mathbf{q}_D = \mathbf{o}$ ). Let the parameters of  $F$  be denoted  $\mathbf{q}_F$ . We can use the Jacobian  $d\mathbf{q}_F/d\mathbf{q}_D$  to find how changes in  $\mathbf{q}_D$  affect  $\mathbf{q}_F$ . But this Jacobian would need to be derived and implemented to interface between every adapter and every `Implicit` primitive and operator in the modeling system.

Equations (2) and (3) provide a more general solution. We construct a collection of control particles and apply the deformation  $D$  to them, which causes the effect of the distortion to be applied to the original parameters of  $F$ .

We create a special array of particles  $\mathbf{p}^i$ . Even though each particle constrains the surface to pass through a three-dimensional point, the constraint restricts only one degree of freedom, since the particle may freely move across the two degrees of freedom along the surface. Hence, the number of particles  $n$  in the array should be  $|\mathbf{q}_F|$ . We assume that  $|\mathbf{q}_D| \ll |\mathbf{q}_F|$  and  $F$  is flexible enough to find the solution of a much less flexible deformation  $D$ .

We then solve a variation of (3) specifically for processing the effect of  $D$  into the parameters of  $F$ ,

$$\sum_j (F_{\mathbf{q}}^i \cdot F_{\mathbf{q}}^j) \lambda^j = F_{\mathbf{x}}^i \cdot (D(\mathbf{p}^i) - \mathbf{p}^i) + \phi(F^i - F(\mathbf{p}_0^i, \mathbf{q}_0)). \quad (15)$$

The resulting Lagrangian multipliers  $\lambda^j$  provide the solution as

$$\dot{\mathbf{q}}_F = - \sum_j \lambda^j F_{\mathbf{q}}^j. \quad (16)$$

We have assumed no desired parameter velocity ( $\dot{\mathbf{Q}} = 0$ ).

Equation 15 has a slightly different feedback term that allows  $F(\mathbf{p}^i)$  to be fixed to an arbitrary value, instead of just zero as was the case in (3). This feedback term makes sure the values at the particles do not drift away from their original values, which are found by evaluating  $F$  at the pre-deformation particle locations  $\mathbf{p}_0^i$ . Hence we can place particles anywhere in space to capture the field of  $F$  instead of just its implicit surface.

We sprinkle these particles randomly in space instead of across the surface. The implicit surface of  $aF$  is the same as that of  $F$  for any  $a \neq 0$ . Using particles on the surface constrained to  $F = 0$  could yield an  $aF$  result with  $a \neq 1$ <sup>2</sup>.

## 5.2 Implementation

We implemented (15) and (16) in the `setq` method of the adapter. When a surface control particle is dragged, (2) and (3) determine new parameters for the adapter's deformation through Euler integration of  $\dot{\mathbf{q}}$ . These new adapter parameters are then set by the particle system calling `setq`.

The `setq` of the adapter performs the following algorithm.

1. Set the state of its deformation  $D$  to use the parameters from the parameter vector  $\mathbf{q}$  passed to it.
2. Use the deformation  $D$  to evaluate (15) and (16) to find the resulting parameter velocity  $\dot{\mathbf{q}}_F$  of its operand  $F$ .
3. Perform an Euler step on the velocity  $\dot{\mathbf{q}}_F$  by adding a fraction of it to the operand's parameter vector returned by `F->getq`, and store the result back in the operand's parameter vector via `F->setq`.

Since the parameters have been passed from the adapter to its operand, the adapter then returns its parameters to their original state. Hence, when an adapter's control particle is moved on an implicit surface, the adapter's parameters remain fixed and its operand's parameters change instead. This is the primary difference between an adapter and an operator in our modeling system.

## 5.3 Results

Figure 3 demonstrates this process on the `Mover` adapter applied to an ellipsoid that was originally placed at the origin. We have placed the translation adapter on the object and dragged the resulting composite object with a single particle. The parameters of the translation are automatically propagated to the parameters of the underlying implicit ellipsoid primitive.

<sup>2</sup>This is also an issue when constructing implicit surfaces using radial basis functions. One or more constraint points are placed inside or outside the desired surface to indicate a desired interior or a desired local surface orientation [13].





The core of this representation was a group project of a class on advanced surface modeling taught in the Fall Semester 2000 at the University of Illinois. Each of the students was assigned a component of the library to implement as a project for the class. This format for a class project had several advantages. The student projects were not discarded at the end of class, which gives the students a stronger sense of accomplishment. The student projects were also distinct, which encouraged cooperation and teamwork instead of competition among the students. The interdependencies among the components of this library meant that it was in a student's best interest to help any other student that might be falling behind. This exercise provided production programming experience in an environment similar to the one many will find in their first jobs.

## 6.2 Future Work

The application of program differentiation tools on a given code segment is itself a complex task. It would be useful to develop a simpler subset of existing program differentiation tools specifically for automatically translating `proc` methods into `grad` and `procq` methods.

The ability to “flatten” chains of multiple operators into a single operators is often employed in other hierarchical models in computer graphics. Depending on the success of implementations on implicit surfaces, it may be interesting to reparameterize the interfaces of other shape representations.

We have implemented the `Mover` adapter to verify the derivations in Section 5. We are in the process of implementing other adapters to perform deformations such as scale, rotation, taper, twist and bend. Barr [2] introduced the latter non-affine deformations, and it will be interesting to see how well some implicits, such as high-degree algebraics, can simulate the deformation effects within their parameterizations.

Implicit surfaces are still *slippery* [14]. We have found that it is much easier to “pull” a convex surface than to “push” it. The *slipperiness* of the surface appears related to the flow gradient of the surface in the direction of the user-exerted force on a control particle. Constraining a control particle to a given position on the surface relative to nearby features could reduce the slippery feel of this method of modeling.

## 6.3 Acknowledgments

The core of our implicit surface library was coded by CS497JCH students Ed Bachta, Lennie Brown, Nate Carr, Jeff Decker, Bill Nagel and Steve Zelinka. Bill Lorensen, Will Schroeder and Ross Whitaker provided valuable insights into object oriented libraries from their experience

with the `vtk` project. This research is supported in part by the NSF grant CCR-0196226 and the University of Illinois Department of Computer Science.

## References

- [1] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, and V. Savchenko. Hyperfun project: a framework for collaborative multidimensional f-rep modeling. *Proc. Implicit Surfaces '99*, pages 59–69, Sept. 1999.
- [2] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3):21–30, July 1984.
- [3] C. H. Bischof, L. Roh, and A. J. Mauer-Oats. ADIC: an extensible automatic differentiation tool for ANSI-C. *Software: Practice and Experience*, 27(12):1427–1456, 1997.
- [4] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.
- [5] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, June 1996.
- [6] C. Hoffman and J. Hopcroft. Automatic surface generation in computer aided design. *Visual Computer*, 1:92–100, 1985.
- [7] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *Visual Computer*, 11:429–446, 1995.
- [8] H. K. Pedersen. `imp`. Source code available via `implicit.eecs.wsu.edu`, 1997.
- [9] W. Schroeder, W. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. *Proc. Visualization '94*, pages 40–45, Oct. 1994.
- [10] W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, Dec. 1997.
- [11] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. *IEEE Visualization '96*, pages 93–100, Oct. 1996.
- [12] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Computer Graphics (Annual Conference Series)*, pages 279–286, Aug. 1997.
- [13] G. Turk and J. O'Brien. Shape transformation using variational implicit functions. *Computer Graphics (Proc. SIGGRAPH 99)*, pages 335–342, Aug. 1999.
- [14] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)*, pages 269–277, July 1994.
- [15] B. Wyvill, E. Galin, and A. Guy. Extending the CSG tree: Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, June 1999.

# A Programmable Particle System Framework For Shape Modeling

Wen Y. Su\*      John C. Hart  
University of Illinois Urbana-Champaign

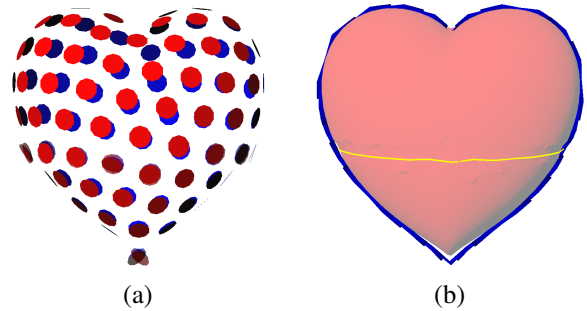
## Abstract

*Particle systems are an effective tool for visualizing information in a variety of contexts. This paper focuses on the use of surface-constrained particles to visualize information about the surface. We have designed a particle system programming framework consisting of behaviors, attributes and shaders that allows users to rapidly create, debug, and deploy particle systems for sensing and extracting specific surface information and displaying this information in an visually effective manner. We also introduce a simple particle system “little language” to facilitate the articulation of these particle programs. We demonstrate the flexibility and power of this framework for surface visualization with the applications of singularity detection and display, non-photorealistic surface illustration, and surface mesh algorithm visualization.*

## 1. Introduction

Particle systems are widely used in many areas of computer graphics and scientific visualization. Among these uses, they have also become an effective tool for the visualization and interactive modeling of surfaces, and there are now a wide variety of different particle systems devised for performing various shape modeling tasks. This paper describes a new particle system programming framework (and accompanying open source library) designed to facilitate the implementation of these systems and the exploration of new particle-based shape modeling applications.

The novel contributions of this systems paper include (1) the decomposition of particle programs into attribute, behavior and shader building blocks, (2) the decomposition of behaviors into force, constraint, integration and cleanup phases, (3) the derivation of new behaviors that cause particles to move to the silhouette or to singularities of implicit surfaces. The remainder of the paper is devoted to motivating and describing the design of this particle system framework and demonstrating its usefulness for rapidly implementing previously published particle systems and exploring new particle system ideas.



**Figure 1. (a) Witkin-Heckbert floater particles. (b) A particle system prototyped in our framework showing the surface polygonized with its silhouette in blue and singularities in yellow.**

Particle systems were first used to model natural phenomena like fire, smoke and water [20]. Though an early application described how an oriented particle system could self-organize to define a surface [24], the idea of using particles to track, display and control a predefined dynamic surface has achieved more success with a wider variety of applications. Witkin and Heckbert [32] showed how a particle system can be used as an effective tool for real-time display and manipulation of implicit surfaces. They constrained a particle system to lie on the surface, displaying it with a textured field of opaque oriented polkadots on an implied clear transparent surface (e.g. Fig 1(a)). This particle system also served as a manipulation widget by selecting some particles and solving a dynamic constraint to force the implicit surface to pass through these particles.

A large number of applications that rely on surface-constrained particle systems have appeared. The mutual point repulsion of surface-constrained particles yeilds a useful method for evenly sampling a surface, and has been used for reaction diffusion texturing [26], retessellating meshes [27], cellular textures [7], local surface parameterization [18], morphing [14], free-form modeling [30, 16, 10] and surface texture synthesis [28, 29], to name a few.

\* Currently at NVidia Corp.

The use of a surface constrained particle system is thus an important tool for computer graphics. However, its varied applications often require sometimes subtle, sometimes major, alterations of (1) the dynamics of particles as they distribute across the surface, (2) the state carried by each particle and the particle system as a whole, and (3) the particle-user communication through appearance and interactivity. Though minor alteration can often be accomplished through parameter changes in an existing particle system, major alteration has often, as reported by many in the literature, required the construction of new particle system applications largely from scratch.

A similar situation occurred in the 1980's when the graphics community aggressively investigated the utility of writing programs that procedurally generated texture and other appearance phenomena, e.g. [4, 19]. New shading systems were reconstructed, largely from scratch, until in 1990 the Renderman shading language [8] showed how a small, focused language could simplify, clarify and standardize the articulation of procedural shading algorithms. Our goal is to provide the same level of assistance to the articulation of particle systems, focused primarily on the particle systems used for shape modeling and texturing.

Fleischer *et al.*[7] took a step in this direction by decomposing particle behaviors into component pieces (e.g. separating interparticle repulsion from surface adhesion), then integrating these pieces through a biological metaphor. Alias's Maya graphics system contains an little language called MEL that includes Particle Expressions capable of customizing the behavior of a particle system, but requires the composition of custom behaviors largely from scratch and lacks the modularization of behaviors and attributes.

Our proposed framework extends Fleischer *et al.*'s behavior decomposition concept into a more complete particle programming framework capable of broader applications than cellular texturing. We also seek the programmability found in MEL's Particle Expressions, though embedded within the modular approach of Fleischer *et al.* we find so attractive.

As described in more detail in Sec. 3, our system organizes a particle system into (1) behavior objects that decompose and compartmentalize the action of a collection of particles into reusable and interchangeable modules, (2) shader objects that likewise encapsulate the appearance and user interactivity of the particles, and (3) attribute objects that contain state information and utilities that can be easily shared and accessed among the different combinations of behavior and shader objects. These objects are collected together to describe a homogeneous collection of particles in a Particles object, and multiple Particles are collected into a ParticleSystem object. For example, the Witkin-Heckbert particle system [32] would consist of two Particles objects: floater particles that disperse across the surface to display

it, as in Fig. 1(a), and controller particles that serve as direct manipulation widgets used to model the surface.

The modular design of this framework provides an intuitive mental model for the rapid prototyping of particle programs from reusable high-level building blocks, while the programmability of the building blocks allows the programmer to add functionality efficiently while focusing only on the added component. This organization resembles that of Renderman, which similarly allows the custom coding of shader building blocks that could be reused in high-level shading networks.

This new framework enables the easy implementation of existing particle systems from their description in the literature and the exploration of new applications. Sec 3 concludes with a demonstration of the Witkin-Heckbert particle system implemented in this framework. We then use the framework for other applications. For example, Fig. 1(b) demonstrates the use of the system to find and visualize the singularities of an algebraic system, described in more detail in Sec 4. We also investigate the use of particle systems for non-photorealistic rendering in Sec. 5, polygonization in Sec. 6 and mesh clustering in Sec. 7.

## 2. Previous Work

Witkin and Heckbert's floater and controller particles [32] inspired the framework described in this paper. They used a surface constrained oriented particle system to display an implicit surface in real-time, faster than possible then by ray tracing or direct polygonization [1, 2]. Their floater particles could be decomposed into three basic behaviors. An adaptive repulsion system centers a variable-width Gaussian energy function at each particle whose gradient exerts a force on neighboring particles in search of an energy minimizing equilibrium. A surface adhesion constraint restricts the force created by repulsion on the particles, by subtracting the force component normal to the surface. Finally, a population control system subdivides isolated particles and deletes crowded particles, and also dynamically grows or shrinks the support of the per-particle Gaussian energy function depending on the proximity of neighboring particles to hasten equilibrium. The main behavior of the controller particles is to convert their velocity to a corresponding change in the implicit surface parameters. The end of Sec 3 describes how our framework implements this particle system.

Our framework more closely resembles that of Fleischer *et al.*[7], which decomposed the Witkin-Heckbert system into component behaviors. Our framework extends this decomposition into behaviors and shaders, the latter of which controls the appearance and user interaction of particles. The Fleischer *et al.* system also assumes a global state which we have further decomposed into shared attribute modules

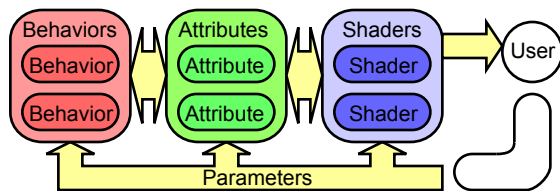
that can be loaded into the particle system on demand. Furthermore, we have subdivided the specification of behaviors into four key phases: force, constraint, integration and cleanup, and interleaved these phases among the behaviors to ensure they occur in the proper order.

An alternative to our particle system programming framework is provided by commercial graphics systems, exemplified by Alias's Maya package. Maya includes MEL a scripting language designed for programming custom behavior that includes Particle Expressions for controlling the behavior of particle systems. These Particle Expressions operate on a global collection of particle attributes that resemble the attributes of our framework except they have not been modularized, and neither have the behaviors and shading instructions specified by the Particle Expressions. While any of our examples could have been implemented in Maya, this lack of clear modularization hinders the ability to reuse individual particle behaviors or rapidly construct applications from a collection of building block modules.

### 3. A Particle System Framework

In the Wickbert system, a *Particles* is a homogeneous collection of particles that behave similarly, and one or more of these can be collected together in a *ParticleSystem* object. A *Particles* object contains a sequence of *ParticleBehavior* objects that collectively describe the action of the particles, a sequence of *ParticleShader* objects that describe how the particles are drawn (and manipulated), and a collection of *ParticleAttribute* objects that contain shared data used by the behaviors and shaders.

Each of the *ParticleAttribute*, *ParticleBehavior* and *ParticleShader* objects inherits the same base class *ParticleStuff* which provides a uniform interface for managing per-object and per-particle parameters. This design allows an application to provide a single dynamic GUI interface to adjust parameters for any of the attributes, behaviors and shaders. Thus new behaviors, shaders and attributes can be designed



**Figure 2. Particles object data flow. Attributes collect information shared by Behaviors and Shaders. User observes particles and interactively adjusts parameters.**

and plugged into the system, and existing ones can be adjusted at run time.

#### 3.1. Attributes

The per-particle and global data elements and procedures of a *Particles* object are organized into several atomic *ParticleAttribute* objects. These attributes are stored and referenced by name in an associative array. Behaviors and shaders access an attribute at an initialization phase by the *attachAttribute(attr;name)* method, which finds the attribute named *name* and assigns the pointer *attr* to it. If no such attribute exists, it finds and assigns the first compatible attribute of the type of the pointer *attr*.

**Examples:** An *ImplicitInterrogator* attribute contains a pointer to an implicit surface object similar to the those described elsewhere [32, 9]. An *AdaptiveRepulsionData* attribute contains the per-particle radii of the particles as well as the global parameters setting the desired and maximum radii values of the particles.

#### 3.2. Behaviors

The motion of particles is represented by an ordered collection of *ParticleBehavior* objects, each of which represents a different atomic component of the behavior of the particles. The contribution of a *ParticleBehavior* to the behavior of a particle is divided into four computational steps:

- *applyForce()* - add this behavior's force to a force accumulator (a per-particle attribute),
- *applyConstraint()* - modify the current force on each particle to satisfy a constraint on its motion,
- *integrate()* - update the particle (e.g. its position, velocity) based on the constrained force held in each particle's force accumulator, and
- *cleanup()* - create and destroy particles based on population dynamics.

These steps are interleaved among all of the behaviors such that the *applyForce()* method of each of the behaviors (in order) is called first, then *applyConstraint()* of each behavior, then *integrate()* and *cleanup()*. Thus the constraint operates on particles after all of the forces have been computed and applied, integrate operates after all forces have been modified by all constraints, and cleanup operates after all particles have updated their new state.

**Examples:** The *applyForce()* method of the *ParticleRepulsion* behavior computes a force that drives particles away from each other, whereas the *applyConstraint()* method of the *SurfaceAdhesion* behavior removes the component of the accumulated force that would cause a particle to leave an implicit surface.

### 3.3. Shaders

An ordered collection of *ParticleShader* objects controls particle appearance and user interaction. The *ParticleShader* object includes a *draw()* method which defaults to calling its *drawParticle(i)* method on each particle *i*. A shader can define a new particle appearance by redefining *drawParticle()*, or can alter a global appearance by redefining *draw()*. Since each shader can change the graphics state, their order is important. We also control user interaction through the shader, by redefining its *event()* method, which is called by the application whenever a user interaction event occurs (e.g. a mouse click).

**Examples:** The shader *ParticleShaderDisk* redefines *drawParticle(i)* to draw a disk in the current coordinate system, which is automatically set up by *ParticleShader* defaults for particle *i* whereas the shader *ParticleShaderConstMaterial* redefines the OpenGL material appearance attributes for all shaders that follow it by redefining its *draw()* method. The shader *CopyParticle* waits for a certain mouse event to occur on one of its particles, and when it does, directs a different *Particles* collection to create a new particle at the current particle's position.

### 3.4. The Programming Framework

We have found that this organization of the state, behavior and appearance of particles provides an intuitive mental model for the articulation of particle systems. It forces the programmer to dissect the conception of a particle system into its component behaviors, the state needed by those behaviors, and how it should communicate the results to the user through its appearance and event processing. This organization also promotes the abstraction of behaviors, attributes and shaders into reusable components that can be connected in a variety of different configurations for the rapid prototyping of new particle systems.

As such, this programming framework serves the same purpose for specifying particle systems as did Renderman for specifying the appearance of scenes. Rather than develop a “little language” such as Renderman does for its shaders, we have opted for the familiarity and speed of the C++ language. This framework provides the details of particle system maintenance allowing the programmer to focus on the kernel of the desired new particle action or state.

This framework also benefits from the object-oriented structure of C++, supporting the inheritance and specialization of attributes, behaviors and shaders. For example, the *ParticleLocality* attribute provides a method for finding the neighbors of a given particle within a given radius, and implements this by a simple linear-time global query of the particles, whereas the *ParticleLocalityGrid* attribute inherits the *ParticleLocality* attribute, but redefines its data

structure with a uniform 3-D voxel grid and its neighbor-finding method with a constant-time grid lookup. Behaviors and shaders that depend on particle neighborhood queries can find an attribute of type *ParticleLocality* and use its interface, even though the actual attribute is the more efficient *ParticleLocalityGrid*. Thus the user can use such modules interchangeably depending on application specific needs ranging from simple proof-of-concept prototypes to efficient production runs, or in other cases for time/space complexity tradeoffs.

The rest of the paper demonstrates the benefits of this framework by showing how it can be used to articulate a variety of different particle systems that would otherwise require a significant amount of effort to implement.

### 3.5. Example: Witkin-Heckbert Particles

Table 1 demonstrates this framework applied to the particle system described by Witkin and Heckbert [32]. They constrained a mutually repulsive particle system to an implicit surface to display the surface. The mutual repulsion of these particles was controlled by a dynamic Gaussian energy function and the width of these Gaussians can vary across particles and over time. These particles would also subdivide when isolated and die when overcrowded.

These surface display particles are described by the “Floaters” collection of particles in Table 1. The *AdaptiveRepulsionData* holds the Gaussian energy function information. The *ImplicitInterrogator* attribute is used to query the implicit surface to which the particles adhere. The *ParticleLocality* attribute contains a *getNeighbors(i, radius)* method that returns all particles within a distance of *radius* from particle *i*. This Witkin-Heckbert particle system was based on viscous dynamics where force equals mass times *velocity*, which is handled within the framework by the *ViscousState* attribute and the *ViscousDynamics* behavior.

The second half of the Witkin-Heckbert approach was to select some particles as “controllers” and perform a minimum work adjustment of the implicit surface parameters to ensure the surface always passed through these particles. When a controller particle is dragged, the implicit surface will dynamically adjust to pass through it and the other controller particles.

We implemented this action with the “Controllers” collection of particles in Table 1. This particle collection is initially empty, and particles are added to it by the *CopyParticle* shader in the “Floaters” collection. This particle collection also does not contain any behaviors with an *integrate()* method. Particles are dragged by the *DragParticle* shader which updates its position and stores the difference as a velocity. The *applyConstraint()* method of the *SurfaceDeformation* behavior then converts this particle velocity into a



---

Particles: “Floaters”

ParticleAttributes:

**AdaptiveRepulsionData**

Per-particle: radius

Global: desired\_rad, max\_rad, repulsion\_amp

**ImplicitInterrogator**

GetImplicit() { *return an implicit* }

**ParticleLocality**

getNeighbors(i,radius) { *return neighbors* }

**ViscousState**

Per-particle 3-vector: x,v

ParticleBehaviors:

**ParticleRepulsion**

attachAttribute(AdaptiveRepulsionData)

applyForce() { *compute repulsion* }

integrate() { *change radius* }

**SurfaceAdhesion**

attachAttribute(ImplicitInterrogator)

applyConstraint() { *remove non-tangent vel.* }

**ViscousDynamics**

attachAttribute(ViscousState)

integrate() {  $x += v * dt$  }

**ParticleFate**

attachAttribute(AdaptiveRepulsionData)

cleanup() { *fission and death* }

ParticleShaders:

**ParticleDisk**

attachAttribute(AdaptiveRepulsionData)

drawParticle(i) { *draw disk* }

**CopyParticle**

event(doubleclick) { *create new*

*particle in “Controllers”* }

Particles: “Controllers”

ParticleAttributes:

**ImplicitInterrogator**

**ViscousState**

ParticleBehaviors:

**SurfaceDeformation**

attachAttribute(ImplicitInterrogator)

attachAttribute(ViscousState)

applyConstraint() { *transfer particle velocity to implicit parameter velocity* }

integrate() { *update implicit params.* }

ParticleShaders:

**ParticleCylinder**

**DragParticle**

**Table 1.** The “Witkin-Heckbert94” particle system.

---

parameter velocity and its integrate() method updates the parameters of the implicit surface indicated by the Implicit-Interrogator.

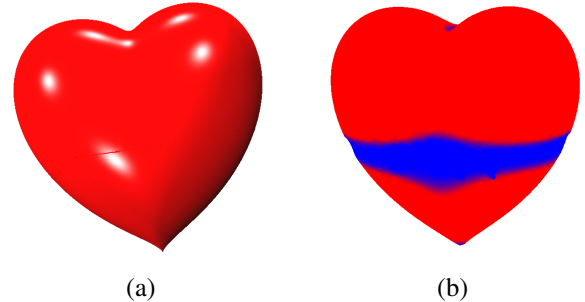
## 4. Singularity Particles

Surface constrained particle systems perform well on smooth surfaces, but can be problematic on surfaces with singularities such as creases, cusps and self-intersections. Such singularities occur on an implicit surface where the gradient vanishes or becomes discontinuous. In such cases, the orientation of particles becomes discontinuous and the particles tend to oscillate about the singularity, as can be seen at the bottom of Fig. 1(a).

Rosch *et al.* [21] first identified this problem, noticing that when two sheets of a surface intersected (e.g. from the 3-D immersion of a Klein bottle) or in areas where curvature increases to infinity (e.g. the cusp of a cone) the particles were sparse. They solved the intersecting-sheets problem by allowing particles to only repel other particles with similar orientation. They overcame the sparseness in high curvature areas by making particle repulsion radius inversely proportional to curvature.

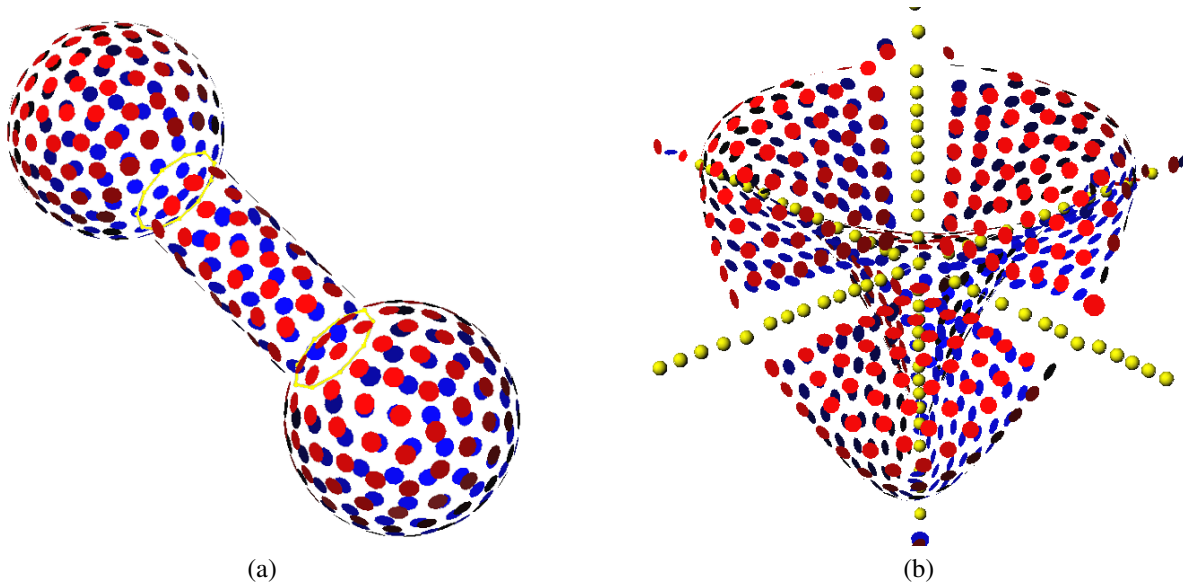
These enhancements could be integrated directly into our framework through an enhanced behavior that inherits ParticleRepulsion but compares particle orientations, and through the addition of a ParticleCurvature attribute that effects the dynamic resizing of the particle radius in the AdaptiveRepulsionData attribute. But we instead propose an alternative solution suited more for the visualization of singularities on an isosurface.

Rather than manipulating the repulsion energy of a particle, we instead use our framework to construct a new collection of particles, called “Singularity Particles”, whose behavior causes them to adhere to the singularities of the surface, and whose shaders indicate they are unoriented with spheres (since the gradient vanishes at a singularity).



**Figure 3.** (a) The crease in the middle of the heart is caused by singularities. (b) Regions with small gradients are indicated in blue.

---



**Figure 4. (a) Singularity particles find features in a CSG model. (b) Singularity particles display degenerate portions of the Steiner surface that ray tracing would miss.**

We will apply these singularity particles to a heart shaped implicit surface [25] of the function

$$f(x, y, z) = (x^2 + 2.25y^2 + z^2 - 1)^3 - x^2z^3 - 0.1125y^2z^3 = 0. \quad (1)$$

This surface contains rather obvious isolated singularities at its bottom and top cusps. What is less obvious is that its gradient magnitude reduces to zero along a closed loop at its midsection, which causes problems with the surface normal when ray tracing it, as shown in Fig. 3. Our goal for singularity particles is to interrogate an implicit surface function to find these problem areas and make them obvious to the investigator.

#### 4.1. Implementation

We implemented singularity particles by creating a new behavior module called *SingularityInterrogator* that adds a force onto particles in the direction opposite to the gradient of the gradient norm squared:  $-\nabla(\|\nabla(f)\|^2)$ . This moves the particle in the direction of decreasing gradient magnitude, in search of a minimum whose value is zero. In other words, we want to move in the direction where the gradient norm squared is small.

The collection of singularity particles is initially empty. We could seed them with particles in random positions across the surface but we found it was better to use the floater particles, which give us the ability to interrogate a sampling of the entire surface, to trigger the creation of new singularity particles. A *DetectSingularity* behavior is added

to the floater particles that tracks the gradient of the implicit surface at the particle’s position. When the gradient’s magnitude falls below a preset threshold, or when its direction changes faster than a present threshold, the behavior directs the “Singularity Particles” to create a new particle at that location. The singularity particles otherwise follow the same behaviors as the floaters (though with the added force of the *SingularityAdhesion*) and subdivide in an effort to grow and cover whatever singularity exists, such as the closed loop at the midsection of the valentine heart.

We also alter the *ParticleLocality* attribute in the “Floaters” particles so its *getNeighbors()* method returns not only nearby particles from “Floaters” but also from “Singularity Particles”. Thus the singularity particles serve as a warning barrier to prevent floater particles from venturing too close to regions of the surface which cause floaters to malfunction.

#### 4.2. Discussion

We found that the combination of sampling with the floaters and searching with the singularity particles converges quickly for many shapes including those surfaces with hard corners. We tested it a number of surfaces including CSG models and the Steiner surface.

The singularity particles also serve as a feature detector as they naturally find creases. The dumbbell shape formed by the union of a capped cylinder with two spheres contains two circular creases at the sphere-cylinder intersections. The floater particles are unstable near these locations,



and so generate singularity particles that find the creases and move the unstable floaters away, as shown in Figure 4(a).

Steiner's surface

$$f(x, y, z) = x^2y^2 + y^2z^2 + z^2x^2 + xyz \quad (2)$$

includes the three coordinate axes, because if any of  $x$ ,  $y$ , or  $z$  is zero then  $f$  is zero. These axes are infinitely thin and a simple ray tracer would likely miss these features. The addition of singularity particles makes clear where these degenerate surface segments lie, as shown in Fig. 4(b). This figure reveals some floater particles (the ones that spawned the initial singularity particles) remain trapped on the axes by the repulsion of the singularity particles, but these floaters will eventually die due to isolation.

Thus with the addition of a single new behavior and some reconnection of modules, we adapted a copy of the floater particles to interrogate the singularities of an implicit surface.

## 5. Silhouette Particles

Silhouette curves are widely used in the illustration of surfaces as an effective method for visually conveying the shape of a surface without the overhead or distraction of photorealistic shading [17, 12, 31, 3]. What we call the silhouette curves are defined as all points on the surface where the normal of the point  $N$  and the view vector  $v$  are perpendicular or  $N \cdot V = 0$ . (This is perhaps more precisely known as the *contour*.) We can use our new framework to construct a new collection of silhouette particles, and use these particles to construct a particle system that yields a non-photorealistic rendering of the implicit surface.

### 5.1. Implementation

Our silhouette particles begin as a copy of the floater particles. To this, we add a new behavior called *SilhouetteAdhesion* that moves the particles toward the silhouette. The silhouette is the zero set of the function  $g(\mathbf{x}) = N \cdot V$  where  $V$  is the view vector. Instead of solving for  $\mathbf{x}$  directly, we use gradient descent search to move particles in the opposite direction of the gradient of  $h(\mathbf{x}) = (N \cdot V)^2$ , the squared magnitude of the silhouette function. This gradient is thus

$$\nabla h(\mathbf{x}) = \nabla(N \cdot V)^2 \quad (3)$$

$$= 2(N \cdot V)((\nabla N)V + N\nabla V). \quad (4)$$

We really care only about the direction of the normal and not that it is unit length, so we can replace  $N$  with  $\nabla f$  and thus the gradient of the non-unit normal  $\nabla N$  is the Hessian matrix of second derivatives  $H$ . The view vector, which depends on  $\mathbf{x}$  and the camera position  $\mathbf{c}$ , is defined as

$V(\mathbf{x}) = \mathbf{c} - \mathbf{x}$  and its gradient is the Hessian  $-I$  which can be simply replaced by the scalar  $-1$ . This yields

$$\mathbf{f} = 2(N \cdot V)(H V - N) \quad (5)$$

as the force  $\mathbf{f}$  implemented in the `applyForce()` method of the new behavior `SilhouetteAdhesion`.

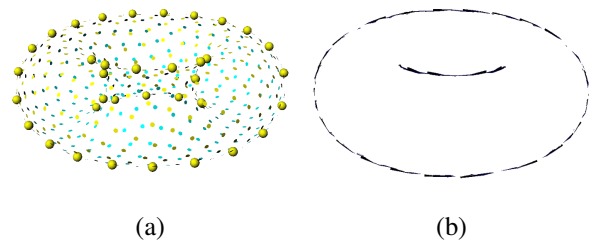
We also created a behavior called `SilhouetteFate` that replaced `ParticleFate` in the silhouette particles. `ParticleFate` was tuned for the distribution of particles on a surface whereas we now need particles to subdivide and populate alive along a space curve.

### 5.2. Discussion

Fig. 5(a) demonstrates the results. We also devised a new shader called *ParticleChain* that finds the two closest particles in the neighborhood and connects them with a line. This provides a nice visualization of the silhouette curves, including its cusps and occluded portions. However, this would not make a very convincing illustration of the torus.

Fig. 5(b) demonstrates the same behavior with different shaders. We modified to `ParticleDisk` shader of the floater particles to draw large overlapping disks of the background color. We deleted the sphere particle shader and altered the `ParticleChain` shader to display more artistic stroke shapes between the silhouette particles. The result is an illustration of the surface that removes the hidden portions of the silhouette curves.

By copying the floater particles, we were able to depend on the fact that our silhouette particles would adhere to the surface and repel each other. This allowed us to focus our effort on the additional force needed to push particles along the surface to the silhouettes.



**Figure 5. Silhouette particles interactively find the silhouette curves on a torus (a). A stroke shader yields a hand drawn look for the same torus, using large floater disks of the background color to hide occluded silhouette segments (b).**

## 6. Dynamic Meshing

While evenly distributed particles allow us to infer a surface, it would be preferable to represent a surface with a triangulation. The speed and robustness of the Witkin Heckbert approach [32] is due partially to their ability to ignore the overhead of connecting particles into a triangle mesh and to maintain the validity of this triangle mesh as the underlying implicit surface changes. The connection of particles into a dynamic triangle mesh has been investigated before (e.g. [23] which also included a topology guarantee). Welch and Witkin used adaptive meshing for free form modeling [30]. Markosian *et al.* used a particle based dynamic mesh that allows users to sculpt free-form surfaces [16]. Here we discuss the implementation of a dynamic mesh using our proposed framework, assuming the per-timestep change in the surface is small and ignoring changes in topology of the underlying surface.

### 6.1. Implementation

Our approach again begins with a copy of floater particles. We augment these particles with a ParticleMesh attribute that encapsulates a half-edge mesh data structure that references particle positions (such as those in ViscousState) as the vertices. A ParticleLocalityMesh attribute inherits and replaces ParticleLocality, and uses the half-edge mesh in the ParticleMesh attribute to accelerate the getNeighbors() function.

The ParticleRepulsion and SurfaceAdhesion behaviors keep the mesh vertices on the surface, but their motion can deform and even invert faces in the mesh. An additional behavior called MeshShape is constructed to perform edge flips in its cleanup() phase in an effort to keep the mesh as close to Delaunay as possible. While some have recommended maximizing the minimum face angle [16], we instead use the criterion of flipping an edge only when the new edge will be shorter. This criterion was easier to implement and appears to work satisfactorily for our example, as shown in Fig. 6.

We also implemented vertex split and edge collapse routines in the cleanup() phase of MeshShape attribute. Some have recommended edge length as a criterion for creation/deletion of new vertices [16], but we have found that setting triangle area and triangle count are also useful factors. We use the Markosian *et al.* [16] idea to sort the potential splits or collapses based on area and only apply to the top portion of them to keep an interactive rendering rate.

### 6.2. Discussion

As before with the shaded strokes of the chain connecting the silhouette particles, we have augmented our parti-

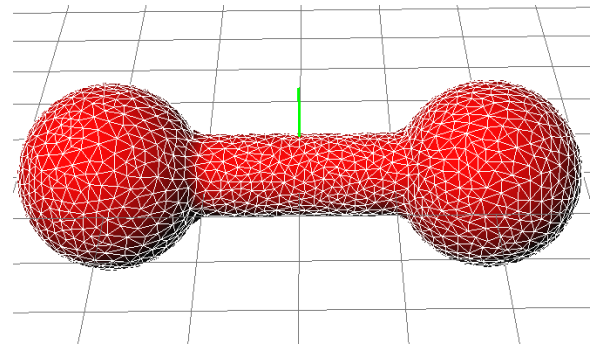


Figure 6. A dynamic mesh.

cle system with an attribute structure whose contents are not necessarily related to a single particle. In this case, the ParticleMesh half-edge data structure does not have a one-to-one correspondence with the particles, which serve as its vertices. Nevertheless, the framework is flexible enough to handle such an attribute.

We also benefit from inheritance by specializing ParticleLocality into ParticleLocalityMesh which uses the mesh datastructure for acceleration. This specialization means we do not need to change ParticleRepulsion, even though the new ParticleLocalityMesh attribute is reporting a different collection of neighbors.

## 7. Mesh Algorithm Visualization

We can use the addition of a mesh in our particle system framework for the application of mesh algorithm visualization. In addition to the ability to implement mesh processing algorithms using our programming paradigm, the execution of these particle programs allows the user to observe the algorithm in action which is useful for the purposes of debugging, presentation and education.

The  $k$ -means algorithm is widely used for clustering data because of its ease of implementation [11], though a variety of variations exist [6, 22, 13, 15]. Since each particle acts independently we can implement cluster growth as a per-particle operation.

### 7.1. Implementation

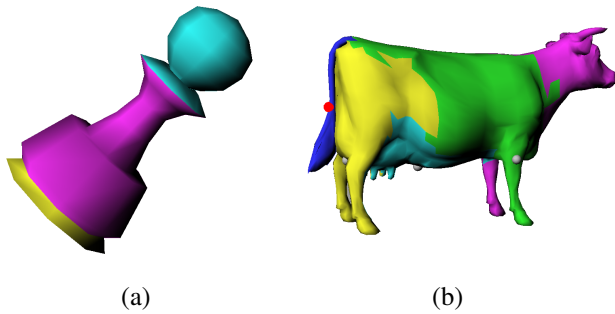
We first create a dynamic mesh collection of particles as specified in the previous section that holds the input mesh. We will call this collection “Mesh.” Even though the mesh is dynamic, it does not change in this example so we delete the ParticleRepulsion behavior. Furthermore, since we are loading a mesh, there is no underlying implicit surface so we delete the ImplicitInterrogator attribute and the SurfaceAdhesion behavior.

We then create a second collection of particles called “Clusters” where each particle will represent a cluster and is positioned at its cluster center, the centroid of its seed face. The “Clusters” particles contain a ClusterMesh behavior that performs the clustering operation. This behavior begins with an initial collection of particles corresponding to the centroids of faces of “Mesh” that represent cluster centers.

The “Clusters” collection of particles also contains a “FaceCluster” attribute which stores a per-particle list of face indices corresponding to faces in the ParticleMesh of “Mesh.” These lists are initialized with the seed face for each particle in “Clusters.”

ClusterMesh proceeds by growing face clusters in the ParticleMesh held in “Mesh.” Once the clusters cover the entire mesh, ClusterMesh moves each of its particles, which correspond to cluster centers, to the centroid of the centermost face of the cluster it generated. ClusterMesh then repeats until the particle positions converge.

A particle shader in “Clusters” is responsible for setting the color of faces in the ParticleMesh attribute of “Mesh” to the correct cluster color. A particle shader in “Mesh” then draws the mesh with its cluster-colored faces and a ParticleSphere particle shader in “Clusters” indicates the location of the cluster centers.



**Figure 7. (a) Pawn model with 254 faces, 3 clusters, 3 iterations. (b) Cow model with 5802 faces, 5 clusters, 10 iterations.**

## 7.2. Discussion

This particle system implementation of  $k$ -means clustering provides an intuitive algorithm visualization, allowing the user to click on any particle to expose the current internal variable settings of its attributes. Users can also interactively steer the partitioning, adding and deleting cluster centers through interaction with the “Cluster” particles.

We use our particle system as a visualization tool in this mesh partitioning algorithm, but the idea of using particle

systems as a debugging tool is not new. Crossno and Angel used this as a software engineering technique [5].

## 8. Conclusion

We have developed a particle system framework and language that allows users to rapidly create and reuse behaviors for a particle system. We have tested this framework with a number of applications that shows the reusability and extensibility due to the design of atomic behaviors.

In applications like singularity searching and artistic rendering for implicit surface, we develop new ways to interrogate properties of implicit surfaces with particles. We also introduce new ways of adapting mesh algorithms in particle systems.

Particle systems are needed to simulate natural phenomena like cloth untangling, water spray, snow, smoke, etc. This idea of iterative refinement implicitly resides in many of these algorithms, which our particle system can exploit. When developing and debugging these algorithms, our particle system helps to visualize each step of the simulation. This provides developers with direct insights to the simulation process. Therefore, our particle system serves as a powerful visualization framework and debugging tool.

### 8.1. Future Work

There are many applications that fit our generalized particle system model. For the applications we have implemented in this paper, for example, dynamic meshing, we would like to update our topology guaranteed polygonization for more complex implicit surfaces. Interactive topology guaranteed polygonization will improve the implicit surface modeling systems. Then we can have real-time polygonization instead of independent particles.

For the particle system itself, we would like to make the particle shading framework more flexible, for example, changing the behavior and shader at run time. This would require run-time interpretation of a particle behavior language. For the moment, we are satisfied with the C++ language for specifying the details of particle attribute, behavior and shader functionality.

In recent years, we have seen rapid advances in the graphics hardware along with graphics programming language e.g vertex and pixel shader. This particle system framework holds promise as a conceptual tool for a possible new way to program vertex shaders.

### 8.2. Acknowledgments

The algebraic heart surface in Fig. 3(a) was ray traced by Lin Shi. This research was supported in part by the NSF ITR CCF-0121288.

## References

- [1] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.
- [2] Jules Bloomenthal and Keith Ferguson. Polygonization of non-manifold implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 309–316. ACM Press, 1995.
- [3] David J. Bremer and John F. Hughes. Rapid approximate silhouette rendering of implicit surfaces. In *Proceedings of Implicit Surfaces 1998*, pages 155–164, jun 1998.
- [4] Robert L. Cook. Shade trees. In *Proc. SIGGRAPH 84*, pages 223–231, 1984.
- [5] Patricia Crossno and Edward Angel. Visual debugging of visualization software: a case study for particle systems. In *Proceedings of the conference on Visualization '99*, pages 417–420. IEEE Computer Society Press, 1999.
- [6] Greg Turk Eugene Zhang, Konstantin Mischaikow. Feature-based surface parameterization and texture mapping. technical report GIT-GVU-03-29, Georgia Institute of Technology, 2003.
- [7] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. In *Proc. SIGGRAPH 95*, pages 239–248, 1995.
- [8] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *Proc. SIGGRAPH 90*, pages 289–298, 1990.
- [9] J.C. Hart, E. Bacht, W. Jarosz, and T. Fleury. Using particles to sample and control more complex implicit surfaces. In *Proc. Shape Modeling International*, pages 129–136, 2002.
- [10] Takeo Igarashi and John F. Hughes. Smooth meshes for sketch-based freeform modeling. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 139–142. ACM Press, 2003.
- [11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [12] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. *ACM Transactions on Graphics*, 22(3):856–861, July 2003.
- [13] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.
- [14] Aaron W. F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *Proc. SIGGRAPH 99*, pages 343–350, 1999.
- [15] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371. ACM Press, 2002.
- [16] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: a constructive approach to modeling free-form shapes. In *Proc. SIGGRAPH 99*, pages 393–400. ACM Press/Addison-Wesley Publishing Co., 1999.
- [17] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 415–420. ACM Press/Addison-Wesley Publishing Co., 1997.
- [18] Hans Kohling Pedersen. Decorating implicit surfaces. In *Proc. SIGGRAPH 95*, pages 291–300, 1995.
- [19] Ken Perlin. An image synthesizer. In *Proc. SIGGRAPH 85*, pages 287–296, 1985.
- [20] W. T. Reeves. Particle systems a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.
- [21] Angela Rosch, Matthias Ruhl, and Dietmar Saupe. Interactive visualization of implicit surfaces with singularities. *Computer Graphics Forum*, 16(5):295–306, 1997.
- [22] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 146–155. Eurographics Association, 2003.
- [23] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 279–286. ACM Press/Addison-Wesley Publishing Co., 1997.
- [24] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 185–194. ACM Press, 1992.
- [25] Gabriel Taubin. An accurate algorithm for rasterizing algebraic curves. In *Proceedings on the second ACM symposium on Solid modeling and applications*, pages 221–230. ACM Press, 1993.
- [26] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proc. SIGGRAPH 91*, pages 289–298, 1991.
- [27] Greg Turk. Re-tiling polygonal surfaces. In *Proc. SIGGRAPH 92*, pages 55–64, 1992.
- [28] Greg Turk. Texture synthesis on surfaces. In *Proc. SIGGRAPH 2001*, pages 347–354, 2001.
- [29] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proc. SIGGRAPH 2001*, pages 355–360, 2001.
- [30] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 247–256. ACM Press, 1994.
- [31] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM Press, 1994.
- [32] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM Press, 1994.